



Building the Next Generation of Parallel Applications

Michael A. Heroux
Scalable Algorithms Department
Sandia National Laboratories

Collaborators:

SNL Staff: [B.|R.] Barrett, E. Boman, R. Brightwell, H.C. Edwards, A. Williams

SNL Postdocs: M. Hoemmen, S. Rajamanickam, M. Wolf

ORNL staff: Chris Baker

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.





Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
5. DRY is not possible across CPUs and GPUs.
6. Extended precision is too expensive to be useful.
7. Resilience will be built into algorithms.
8. GPUs are a harbinger of CPU things to come.
9. Fortran Developers are in trouble in a manycore world.
10. Global SIMD is sufficient parallelism for scientific computing.

Trilinos Contributors

Current Contributors

Chris Baker

Ross Bartlett

Pavel Bochev

Erik Boman

Lee Buermann

Todd Coffey

Eric Cyr

David Day

Karen Devine

Clark Dohrmann

David Gay

Glen Hansen

David Hensinger

Mike Heroux

Mark Hoemmen

Russell Hooper

Jonathan Hu

Sarah Knepper

Patrick Knupp

Joe Kotulski

Jason Kraftcheck

Rich Lehoucq

Nicole Lemaster

Kevin Long

Karla Morris

Chris Newman

Kurtis Nusbaum

Ron Oldfield

Mike Parks

Roger Pawlowski

Brent Perschbacher

Kara Peterson

Eric Phipps

Siva Rajamanickam

Denis Ridzal

Lee Ann Riesen

Damian Rouson

Andrew Salinger

Nico Schlömer

Chris Siefert

Greg Sjaardema

Bill Spotz

Heidi Thornquist

Ray Tuminaro

Jim Willenbring

Alan Williams

Michael Wolf

Past Contributors

Paul Boggs

Jason Cross

Michael Gee

Esteban Guillen

Bob Heaphy

Ulrich Hetmaniuk

Robert Hoekstra

Vicki Howle

Kris Kampshoff

Tammy Kolda

Joe Outzen

Mike Phenow

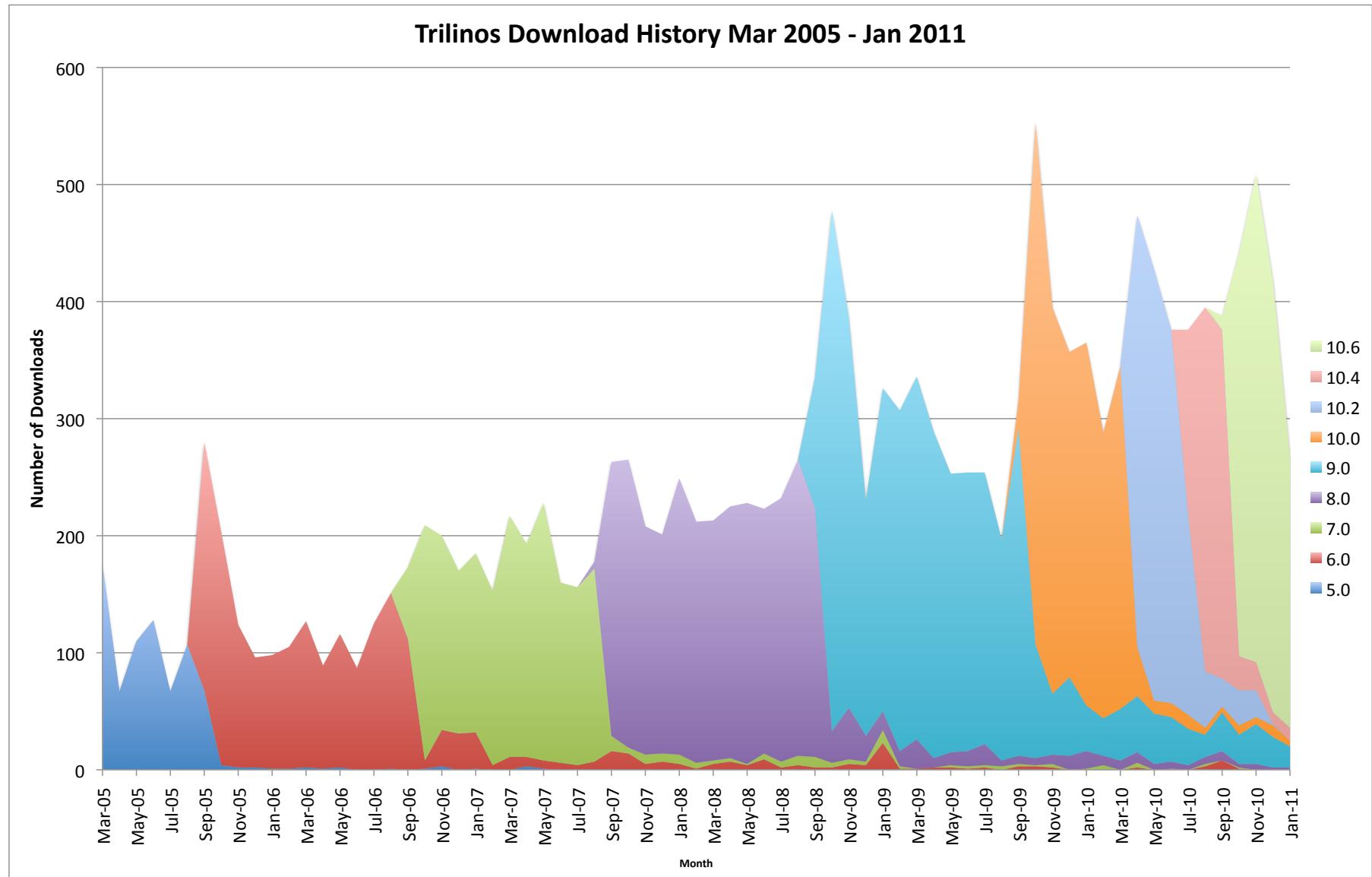
Paul Sexton

Ken Stanley

Marzio Sala

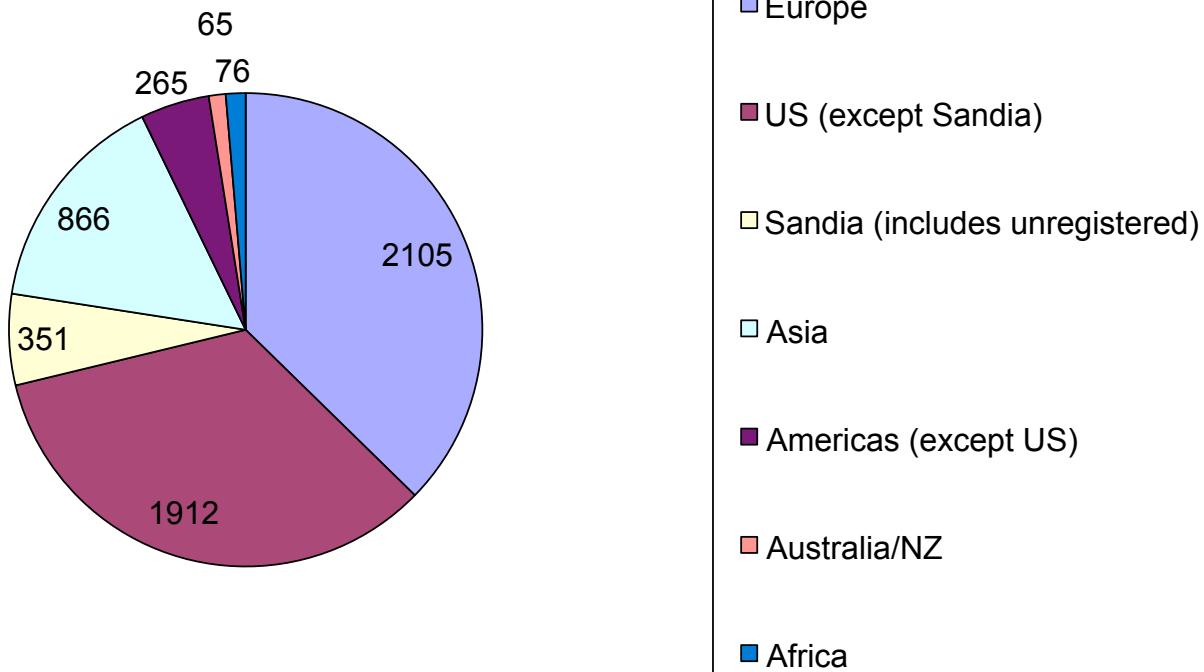
Cedric Chevalier

Trilinos Download History: 17600 Total

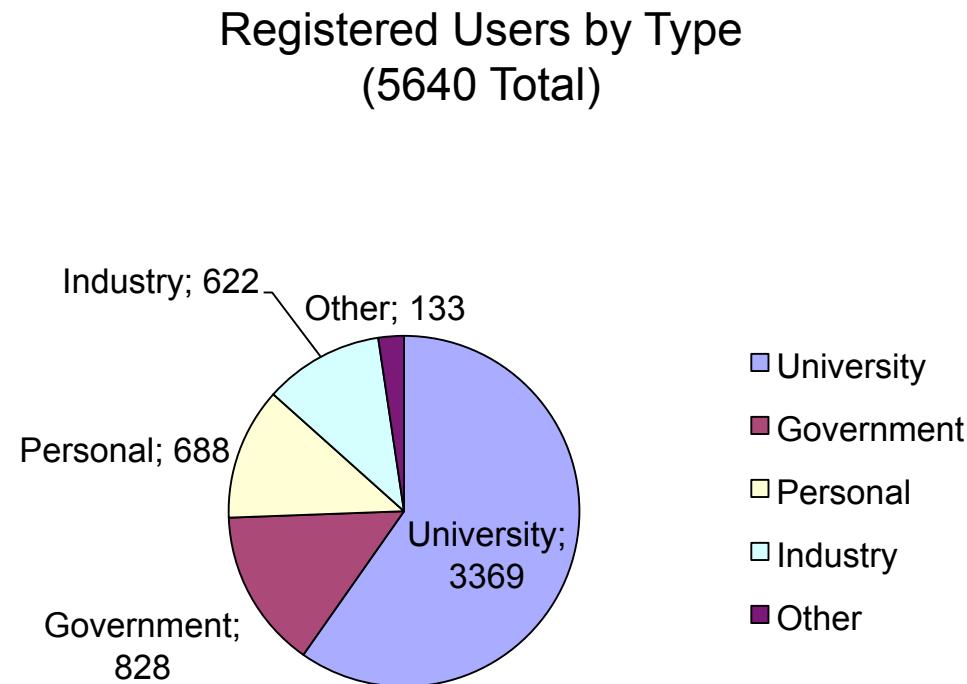


Registered User by Region

Registered Users by Region (5640 Total)



Registered Users by Type



Ubuntu/Debian: Other sources

The following binary packages are built from this source package:

- [libtrilinos](#)
parallel solver libraries within an object-oriented software framework
- [libtrilinos-dbg](#)
parallel solver libraries within an object-oriented software framework
- [libtrilinos-dev](#)
parallel solver libraries within an object-oriented software framework
- [libtrilinos-doc](#)
parallel solver libraries within an object-oriented software framework
- [python-pytrilinos](#)
parallel solver libraries within an object-oriented software framework

Other Packages Related

| | |
|--|--|
| build-depends | build-depends-indirect |
| cdbs common build system for Debians | |
| quilt Tool to work with series of patches | |
| debhelper (>= 7) | |

The following binary packages are built from this source package:

- [libtrilinos](#)
parallel solver libraries within an object-oriented software framework
- [libtrilinos-dbg](#)
parallel solver libraries within an object-oriented software framework
- [libtrilinos-dev](#)
parallel solver libraries within an object-oriented software framework

Links for trilinos

Debian Resources:

- [Bug Reports](#)
- [Developer Information \(PTS\)](#)

```
maherou@jaguar13:/ccs/home/maherou> module avail trilinos
-----
/opt/cray/modulefiles
trilinos/10.0.1(default) trilinos/10.2.0

-----
/sw/xt5/modulefiles
trilinos/10.0.4 trilinos/10.2.2 trilinos/10.4.0 trilinos/8.0.3 trilinos/9.0.2

-----
python-central
register and build utility for Python

libopenmpi-dev
high performance message passing library – header files

libsuperlu3-dev
Direct solution of large, sparse systems of linear equations
```

Download trilinos

Capability Leaders: Layer of Proactive Leadership

- Areas:
 - ◆ Framework, Tools & Interfaces (J. Willenbring).
 - ◆ Software Engineering Technologies and Integration (R. Bartlett).
 - ◆ Discretizations (P. Bochev).
 - ◆ Geometry, Meshing & Load Balancing (K. Devine).
 - ◆ Scalable Linear Algebra (M. Heroux).
 - ◆ Linear & Eigen Solvers (J. Hu).
 - ◆ Nonlinear, Transient & Optimization Solvers (A. Salinger).
 - ◆ **Scalable I/O: (R. Oldfield)**
- Each leader provides strategic direction across all Trilinos packages within area.

Trilinos Package Summary

| | Objective | Package(s) |
|-----------------|--------------------------------|---|
| Discretizations | Meshering & Discretizations | STKMesh, Intrepid, Pamgen, Sundance, ITAPS, Mesquite |
| | Time Integration | Rythmos |
| Methods | Automatic Differentiation | Sacado |
| | Mortar Methods | Moertel |
| Services | Linear algebra objects | Epetra, Jpetra, Tpetra, Kokkos |
| | Interfaces | Thyra, Stratimikos, RTOp, FEI, Shards |
| | Load Balancing | Zoltan, Isorropia |
| | “Skins” | PyTrilinos, WebTrilinos, ForTrilinos, Ctrilinos, Optika |
| | C++ utilities, I/O, thread API | Teuchos, EpetraExt, Kokkos , Triutils, ThreadPool, Phalanx |
| Solvers | Iterative linear solvers | AztecOO, Belos, Komplex |
| | Direct sparse linear solvers | Amesos, Amesos2 |
| | Direct dense linear solvers | Epetra, Teuchos, Pliris |
| | Iterative eigenvalue solvers | Anasazi, Rbgen |
| | ILU-type preconditioners | AztecOO, IFPACK, Ifpack2 |
| | Multilevel preconditioners | ML, CLAPS |
| | Block preconditioners | Meros, Teko |
| | Nonlinear system solvers | NOX, LOCA |
| | Optimization (SAND) | MOOCHO, Aristos, TriKota, Globipack, Optipack |
| | Stochastic PDEs | Stokhos |



Three Design Points

- Terascale Laptop: Uninode-Manycore
- Petascale Deskside: Multinode-Manycore
- Exascale Center: Manynode-Manycore

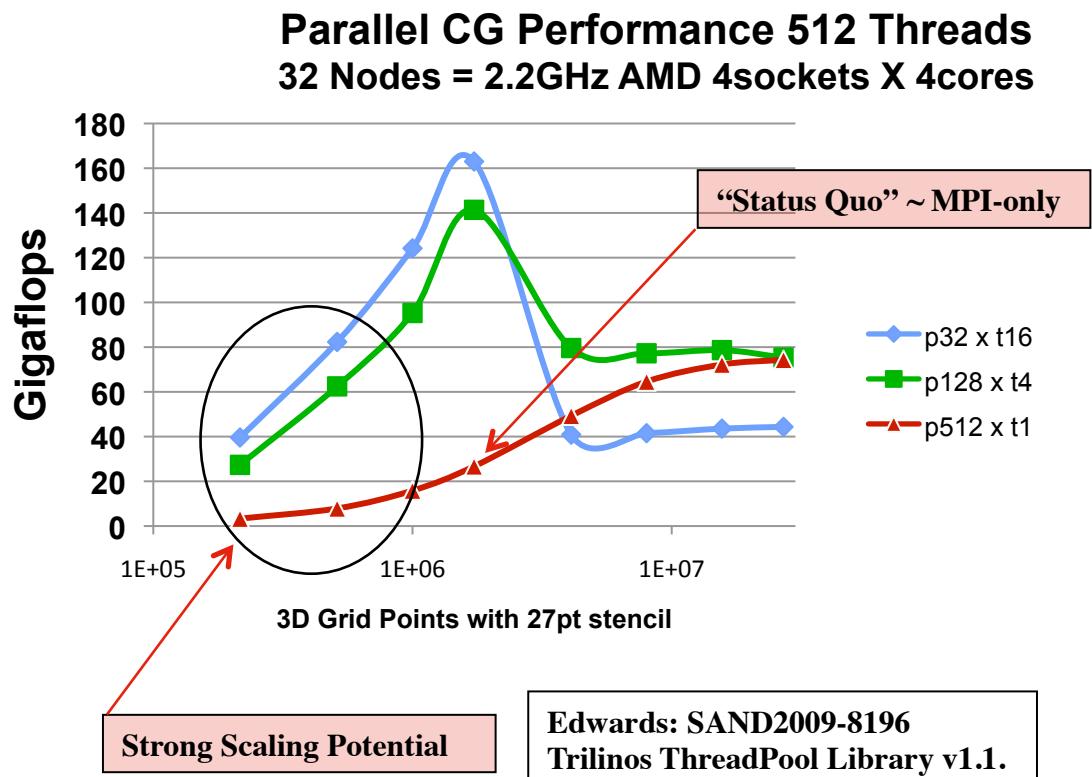


Basic Concerns: Trends, Manycore

- Stein's Law: *If a trend cannot continue, it will stop.*

Herbert Stein, chairman of the Council of Economic Advisers under Nixon and Ford.

- Trends at risk:
 - Power.
 - Single core performance.
 - Node count.
 - Memory size & BW.
 - Concurrency expression in existing Programming Models.
 - Resilience.





Observations

- MPI-Only is not sufficient, except ... much of the time.
- Near-to-medium term:
 - MPI+[OMP|TBB|Pthreads|CUDA|OCL|MPI]
 - Long term, too?
- Concern:
 - Best hybrid performance: 1 MPI rank per UMA core set.
 - UMA core set size growing slowly → Lots of MPI tasks.
- Long- term:
 - Something hierarchical, global in scope.
- Conjecture:
 - Data-intensive apps need non-SPDM model.
 - Will develop new programming model/env.
 - Rest of apps will adopt over time.
 - Time span: 10-20 years.



What Can we Do Right Now?

- Study why MPI was successful.
- Study new parallel landscape.
- Try to cultivate an approach similar to MPI (and others).



MPI Impresssions

MPI: It Hurts So Good

- **Observations**

- “assembly language” of parallelism
 - lowest common denominator
 - portable across architectures
 - upfront effort
 - system
 - environment

• Cr



- lowest common denominator
 - portable across architectures
- upfront effort required
 - system
 - environment
- C++
 - $n = 0$
 - $n = 1$
 - $n > 1$

So What Would Life Be Like Without MPI?

```
long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    x = cilk_spawn fib_parallel(n-1);
    y = fib_parallel(n-2);
    cilk_sync;
    return (x + y);
}
```

```
def fib_parallel(n):
    if n < 2:
        return n
    else:
        return fib_parallel(n-1) + fib_parallel(n-2)
```

The diagram illustrates the recursive definition of the Fibonacci sequence and its parallelization:

Recursive Definition:

$$F(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ F(n-1) + F(n-2) & n > 1 \end{cases}$$

Implementation (Serial C):

```
long fib_serial(long n)
{
    if (n < 2) return n;
    return fib_serial(n-1) + fib_serial(n-2);
}
```

Implementation (OpenMP 3.0):

```
parallel (long n)
  shared(x, n)
  {
      x = fib_sync();
  }
}
```

```
long fib_parallel(long n)
{
    long x, y;
    if (n < 2) return n;
    #pragma omp task default(none) shared(x,y)
    {
        x = fib_parallel(n-1);
        y = fib_parallel(n-2);
        #pragma omp taskwait
        return (x+y);
    }
}
```

Tim Stitts, CSCS

SOS14 Talk

March 2010

Dan Reed, Microsoft

Workshop on the Road Map for the Revitalization of High End Computing

June 16-18, 2003

The image is a collage of various elements. At the top right is a red banner with white text: "Revitalization of High End Computing" and "June 16-18, 2003". Below this is a diagonal line of text: "Looking Forward to a New Age of Large-Scale Parallel Programming and the Demise of MPI" followed by "...hopes and dreams of an HPC educator". To the left of this text is a small orange square containing the letters "CSCS". In the bottom left corner is a cartoon illustration. It features a green T-Rex wearing a yellow "Fortran" shirt. The T-Rex is looking up at a volcano that is erupting black smoke. A speech bubble from the volcano contains the text: "RELAX. WE'RE TOO BIG TO FAIL.". In the background of the illustration are green trees and a blue sky. The overall theme of the collage is the evolution of parallel computing paradigms, specifically the shift away from MPI.

“ MPI is often considered the “portable assembly language” of parallel computing, ... ”

Brad Chamberlain, Cray, 2000.



3D Stencil in NAS MG

HPCs

```

subroutine commut3( axis,n1,n2,n3,kk )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3, k, ierr
double precision u( n1, n2, n3 )
integer i3, i2, i1, buff_len,buff_id
buff_id = 2 + dir
buff_len = 0

if( axis .eq. 1 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1-1, i2,i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2,i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 2 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2-1, i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2,i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 3 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2, i3-1 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2, i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 1,2,3 ) then
  call sync_all()
endif
call zero( u,n1,n2,n3 )
endif
return
end

subroutine give3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3, k, ierr
double precision u( n1, n2, n3 )
integer i3, i2, i1, buff_len,buff_id
buff_id = 2 + dir
buff_len = 0

if( axis .eq. 1 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1-1, i2,i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2,i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 2 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2-1, i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2,i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 3 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2, i3-1 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( n1, i2, i3 )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 1,2,3 ) then
  call sync_all()
endif
call zero( u,n1,n2,n3 )
endif
return
end

subroutine take3( axis, dir, u, n1, n2, n3 )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )
integer buff_id, indx
integer i3, i2, i1
buff_id = 3 + dir
indx = 0

if( axis .eq. 1 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        indx = indx + 1
        u( n1-1, i2,i3 ) = buff(indx, buff_id )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        indx = indx + 1
        u( n1, i2,i3 ) = buff(indx, buff_id )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 2 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        indx = indx + 1
        u( n1, i2-1, i3 ) = buff(indx, buff_id )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        indx = indx + 1
        u( n1, i2, i3 ) = buff(indx, buff_id )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 3 ) then
  if( dir .eq. -1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        indx = indx + 1
        u( n1, i2, i3-1 ) = buff(indx, buff_id )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  else if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        indx = indx + 1
        u( n1, i2, i3 ) = buff(indx, buff_id )
      enddo
    enddo
    buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
      buff(1:buff_len,buff_id)
  endif
endif
if( axis .eq. 1,2,3 ) then
  call sync_all()
endif
call zero( u,n1,n2,n3 )
endif
return
end

subroutine commpl( axis, u, n1, n2, n3, kk )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3, k, ierr
integer i3, i2, i1, buff_len,buff_id
buff_id = 3 + dir
buff_len = nm2

if( axis .eq. 1 ) then
  do i3=2,n3-1
    do i2=2,n2-1
      do i1=1,n1-1
        indx = indx + 1
        u( n1-1, i2,i3 ) = buff(indx, buff_id )
      enddo
    enddo
  enddo
else if( axis .eq. 2 ) then
  do i3=2,n3-1
    do i2=2,n2-1
      do i1=1,n1-1
        indx = indx + 1
        u( n1, i2-1, i3 ) = buff(indx, buff_id )
      enddo
    enddo
  enddo
else if( axis .eq. 3 ) then
  do i3=2,n3-1
    do i2=2,n2-1
      do i1=1,n1-1
        indx = indx + 1
        u( n1, i2, i3-1 ) = buff(indx, buff_id )
      enddo
    enddo
  enddo
endif
if( axis .eq. 1,2,3 ) then
  call sync_all()
endif
call zero( u,n1,n2,n3 )
endif
return
end

subroutine commr( axis, u, n1, n2, n3, kk )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3, k, ierr
integer i3, i2, i1, buff_len,buff_id
buff_id = 3 + dir
buff_len = nm2

if( axis .eq. 1 ) then
  do i3=2,n3-1
    do i2=2,n2-1
      do i1=1,n1
        indx = indx + 1
        u( n1, i2,i3 ) = buff(indx, buff_id )
      enddo
    enddo
  enddo
else if( axis .eq. 2 ) then
  do i3=2,n3-1
    do i2=2,n2-1
      do i1=1,n1
        indx = indx + 1
        u( n1, i2-1, i3 ) = buff(indx, buff_id )
      enddo
    enddo
  enddo
else if( axis .eq. 3 ) then
  do i3=2,n3-1
    do i2=2,n2-1
      do i1=1,n1
        indx = indx + 1
        u( n1, i2, i3-1 ) = buff(indx, buff_id )
      enddo
    enddo
  enddo
endif
if( axis .eq. 1,2,3 ) then
  call sync_all()
endif
call zero( u,n1,n2,n3 )
endif
return
end

```



MPI Reality

Tramonto WJDC Functional

- New functional.
 - Bonded systems.
 - 552 lines C code.

WJDC-DFT (Werthim, Jain, Dominik, and Chapman) theory for bonded systems. (*S. Jain, A. Dominik, and W.G. Chapman. Modified interfacial statistical associating fluid theory: A perturbation density functional theory for inhomogeneous complex fluids. J. Chem. Phys., 127:244904, 2007.*) Models stoichiometry constraints inherent to bonded systems.

How much MPI-specific code?

`dft_fill_wjdc.c`
MPI-specific
code

MFIX

Source term for pressure correction

source_pp_g.f

```

C
Module name: SOURCE_Py_dA_m_n_B_MMIX [HE]
Py_Driver: Py_dA_m_n_B_MMIX [HE]                                     C
C
| correction equation. The off-diagonal coefficients are
| positive. The center coefficient and the source vector are
| positive.
| See copy_Py_dA_m_n_B_MMIX [HE]                                     C
Author: M. Sytaniak                                         Date: 21/09/96 C
Reviewer:                                                 Date: C
C
| Literature/Document References:                                     C
C
Variables referenced:                                              C
Variables modified:                                                 C
C
Local variables:                                                 C
C
^*_C
SUBROUTINE SOURCE_Py_dA_m_n_B_MMIX [HE]
...Translated by BORIS Research VAST-92.0.605 12:17:31 12/09/98
Switches: df
C
| Include parameter file to specify parameter values
| _____
| Modules
C
USE parin
USE parout
USE parerr
USE parrel
USE matrix
USE memory
USE memoryg
USE dafar
USE dafar
USE run
USE run
USE geometry
USE iteration
USE pgeon
USE usrc
USE wshar

```

```

Use array
USE computer
USE float
IMPLICIT NONE
=====
! Global Parameters
=====
! Dummy Arguments
=====
      ! Error index
      INTEGER, INTENT(OUT) :: IER
      ! Septagonal matrix A_m
      DOUBLE PRECISION, DIMENSION(1:m,1:m) :: A_m
      ! Double precision x_0
      DOUBLE PRECISION, DIMENSION(1:m,DIMENSION_M) :: X_0
      ! Vector b_n
      DOUBLE PRECISION, DIMENSION(1:m,DIMENSION_N) :: B_n
      ! Matrix with m rows & n columns
      DOUBLE PRECISION, DIMENSION(1:m,DIMENSION_N) :: M
      ! Convolution weighting factors
      DOUBLE PRECISION XKL_M, XKL_N, XKL_0, XKL_1, XKL_2, XKL_3
      ! XKL(DIMENSION_M,3)
      ! under relaxation factor for pressure
      DOUBLE PRECISION F
      ! terms of km expression
      DOUBLE PRECISION km, kmt, kmr, kmr_t, kmr_t_t, kmr_t_r, kmr_t_rr
      ! Indices
      INTEGER :: L, L1, L2, IMR, IPK, IPK1, IPK2, IPK3, IPK4, IPK5, IPK6
      INTEGER :: M, M1, M2, M3, M4, M5, M6, M7, M8, M9, M10
      ! integer iter
      ! error message
      CHARACTER(80) :: LINE1
      ! FOR CALL_M and CALL_DAT = .true.

```

- MPI-callable, OpenMP-enabled.
 - 340 Fortran lines.
 - No MPI-specific code.
 - Ubiquitous OpenMP markup (red regions).

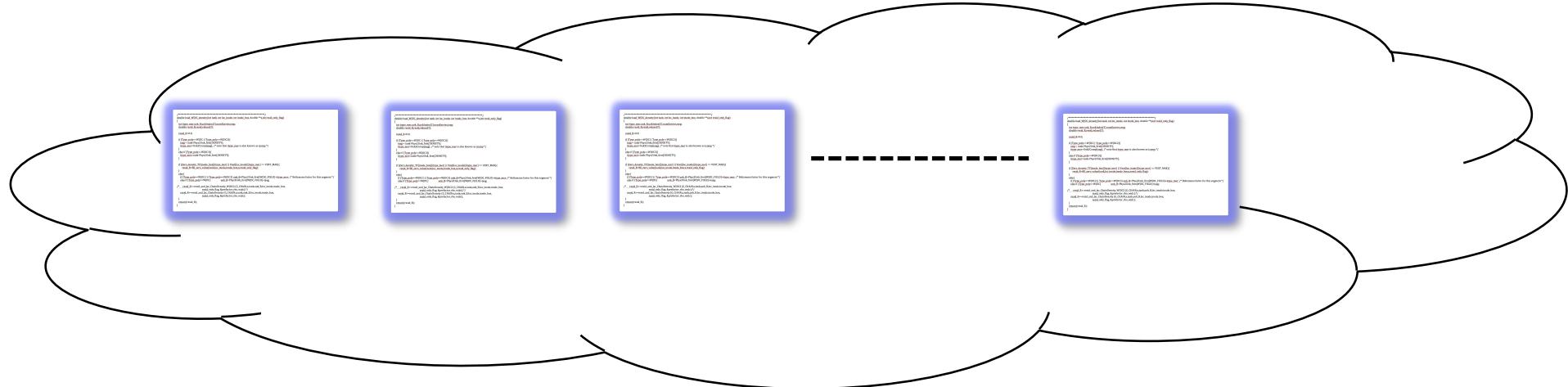


Reasons for MPI Success?

- Portability? Yes.
- Standardized? Yes.
- Momentum? Yes.
- Separation of many Parallel & Algorithms concerns? Big Yes.
- Once framework in place:
 - Sophisticated physics added as serial code.
 - Ratio of science experts vs. parallel experts: 10:1.
- Key goal for new parallel apps: Preserve this ratio

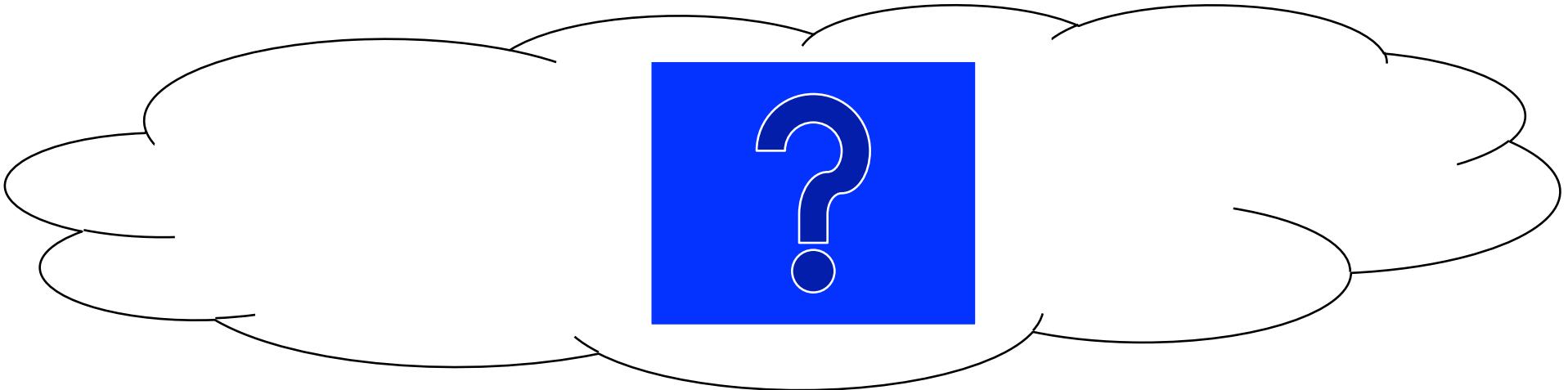


Computational Domain Expert Writing MPI Code





Computational Domain Expert Writing Future Parallel Code





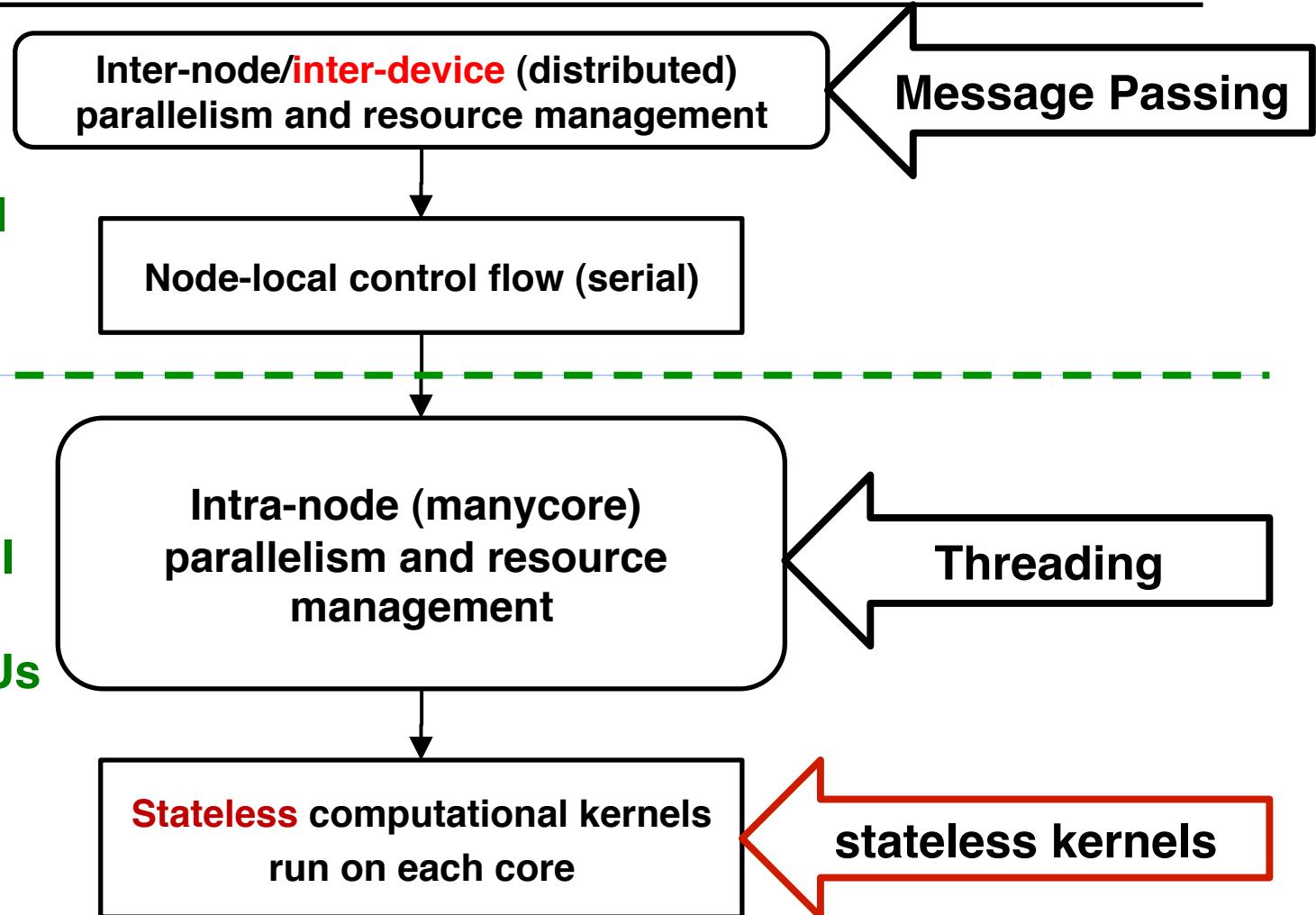
Evolving Parallel Programming Model



Parallel Programming Model: Multi-level/Multi-device

network of computational nodes

computational node with manycore CPUs and / or GPGPU





Domain Scientist's Parallel Palette

- MPI-only (SPMD) apps:
 - Single parallel construct.
 - Simultaneous execution.
 - Parallelism of even the messiest serial code.
- MapReduce:
 - Plug-n-Play data processing framework - 80% Google cycles.
- Pregel: Graph framework (other 20%)
- Next-generation PDE and related applications:
 - Internode:
 - MPI, yes, or something like it.
 - Composed with intranode.
 - Intranode:
 - Much richer palette.
 - More care required from programmer.
- What are the constructs in our new palette?



Obvious Constructs/Concerns

- Parallel for:
 - No loop-carried dependence.
 - Rich loops.
 - Use of shared memory for temporal reuse, efficient device data transfers.
- Parallel reduce:
 - Couple with other computations.
 - Concern for reproducibility.

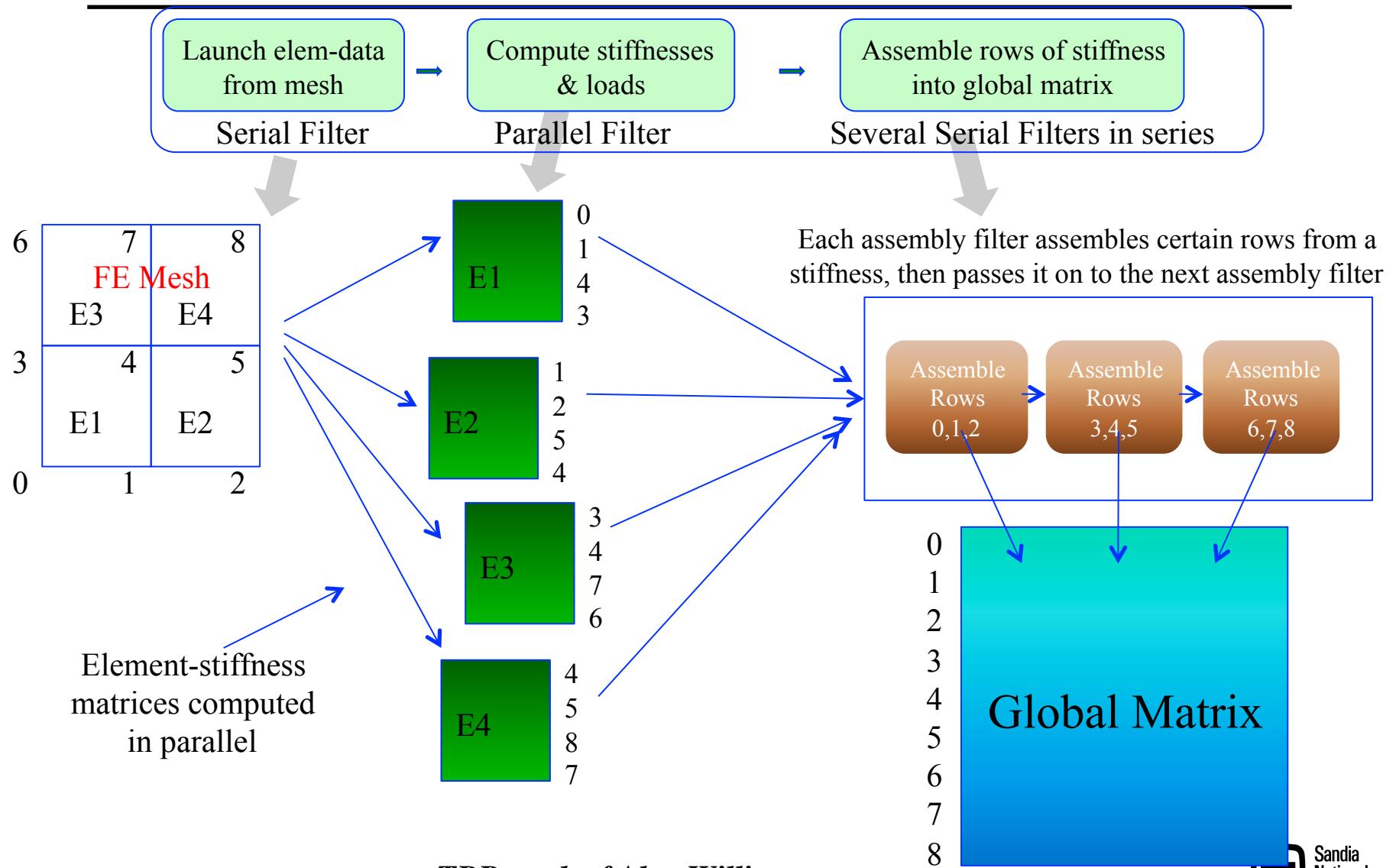


Other construct: Pipeline

- Sequence of filters.
- Each filter is:
 - Sequential (grab element ID, enter global assembly) or
 - Parallel (fill element stiffness matrix).
- Filters executed in sequence.
- Programmer's concern:
 - Determine (conceptually): Can filter execute in parallel?
 - Write filter (serial code).
 - Register it with the pipeline.
- Extensible:
 - New physics feature.
 - New filter added to pipeline.

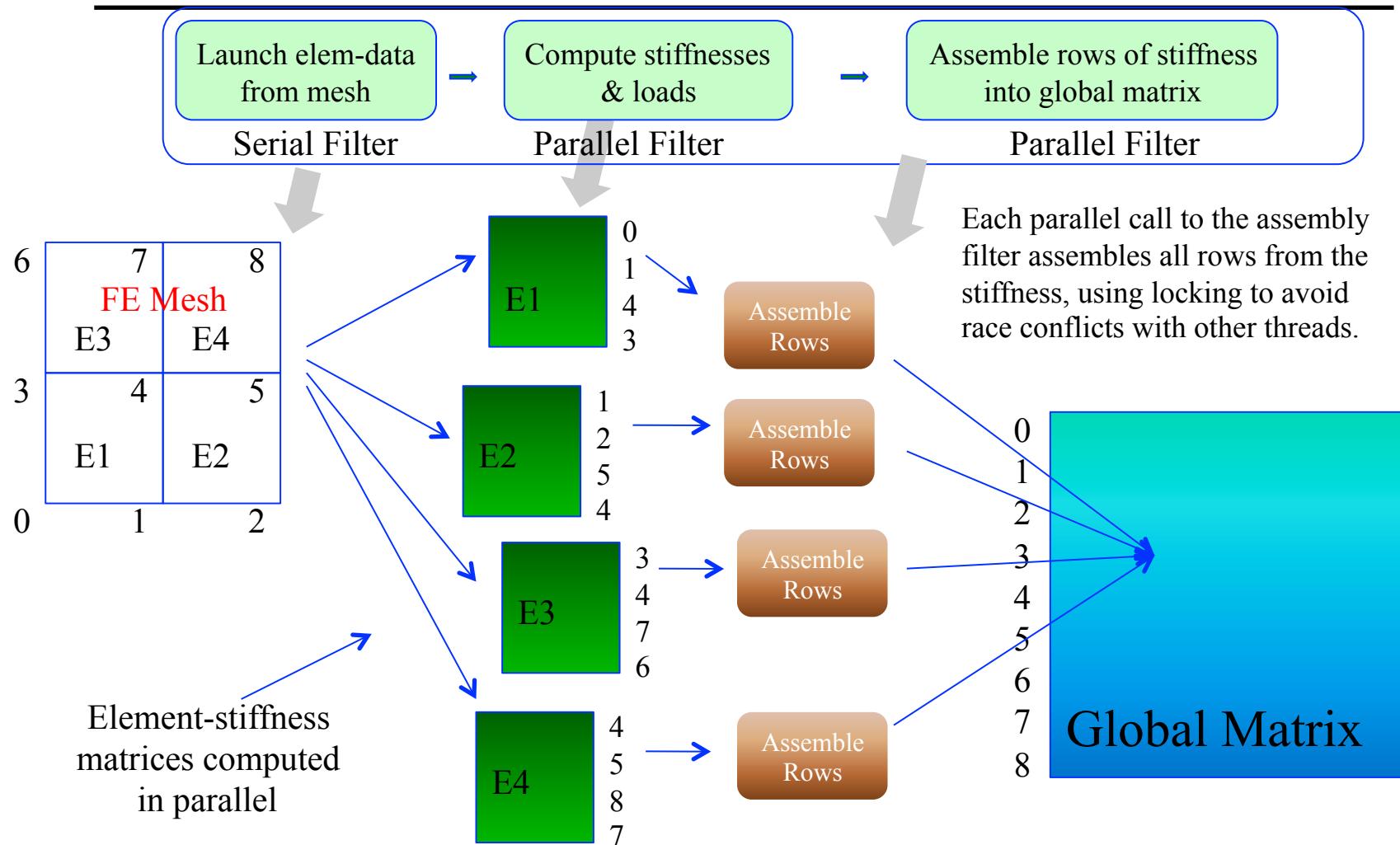


TBB Pipeline for FE assembly





Alternative TBB Pipeline for FE assembly





Base-line FE Assembly Timings

Problem size: $80 \times 80 \times 80 == 512000$ elements, 531441 matrix-rows

The finite-element assembly performs 4096000 matrix-row sum-into operations (8 per element) and 4096000 vector-entry sum-into operations.

MPI-only, no threads. Linux dual quad-core workstation.

| Num-procs | Assembly-time Intel 11.1 | Assembly-time GCC 4.4.4 |
|-----------|-----------------------------|----------------------------|
| 1 | 1.80s | 1.95s |
| 4 | 0.45s | 0.50s |
| 8 | 0.24s | 0.28s |

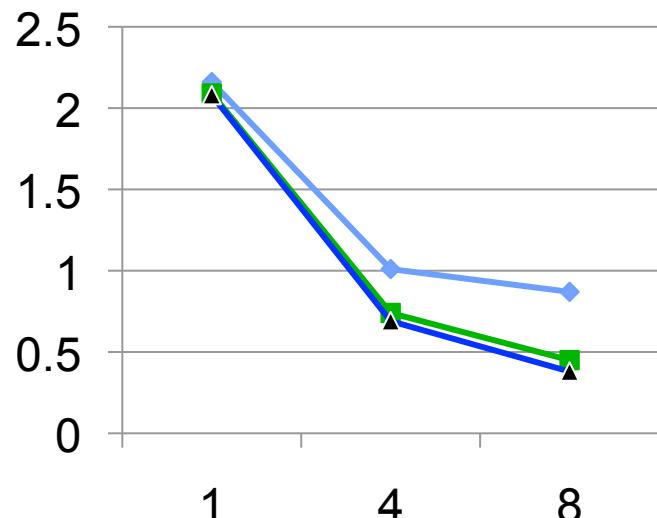


FE Assembly Timings

Problem size: $80 \times 80 \times 80 == 512000$ elements, 531441 matrix-rows

The finite-element assembly performs 4096000 matrix-row sum-into operations (8 per element) and 4096000 vector-entry sum-into operations.

No MPI, only threads. Linux dual quad-core workstation.



| Num-threads | Elem-group-size | Matrix-conflicts | Vector-conflicts | Assembly-time |
|-------------|-----------------|------------------|------------------|---------------|
| 1 | 1 | 1 | 0 | 2.16s |
| 4 | 1 | 4 | 0 | 2.09s |
| 8 | 1 | 8 | 0 | 2.08s |
| 4 | 1 | 95917 | 959 | 1.01s |
| 4 | 4 | 7938 | 25 | 0.74s |
| 4 | 8 | 3180 | 4 | 0.69s |
| 8 | 1 | 64536 | 1306 | 0.87s |
| 8 | 4 | 5892 | 49 | 0.45s |
| 8 | 8 | 1618 | 1 | 0.38s |

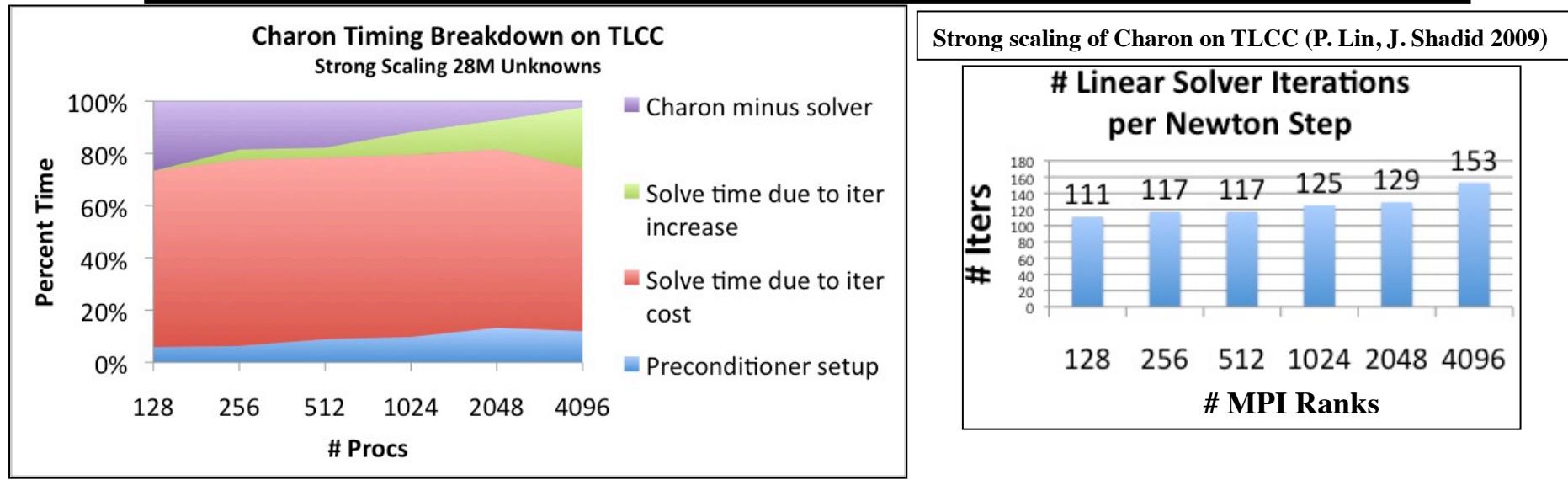


Other construct: Thread team

- Multiple threads.
- Fast barrier.
- Shared, fast access memory pool.
- Example: Nvidia SM
- X86 more vague, emerging more clearly in future.



Preconditioners for Scalable Multicore Systems

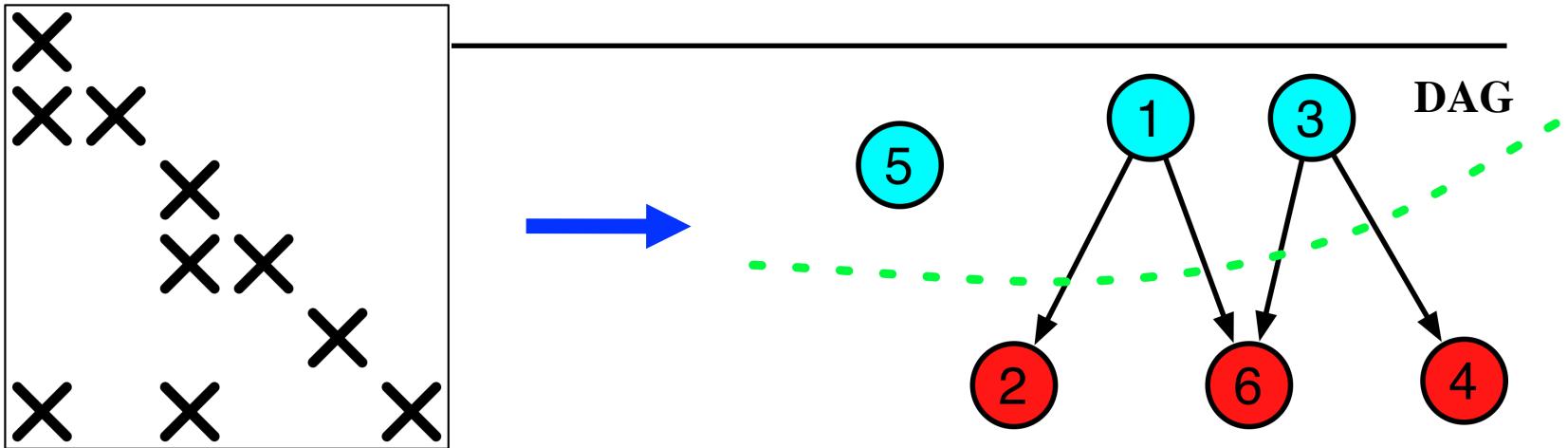


- Observe: Iteration count increases with number of subdomains.
- With scalable threaded smoothers (LU, ILU, Gauss-Seidel):
 - Solve with fewer, larger subdomains.
 - Better kernel scaling (threads vs. MPI processes).
 - Better convergence, More robust.
- Exascale Potential: Tiled, pipelined implementation.
- **Three efforts:**
 - Level-scheduled triangular sweeps (ILU solve, Gauss-Seidel).
 - **Decomposition by partitioning**
 - Multithreaded direct **factorization**

| MPI Tasks | Threads | Iterations |
|-----------|---------|------------|
| 4096 | 1 | 153 |
| 2048 | 2 | 129 |
| 1024 | 4 | 125 |
| 512 | 8 | 117 |
| 256 | 16 | 117 |
| 128 | 32 | 111 |



Level Set Triangular Solver



L

Triangular Solve:

- Critical Kernel
 - MG Smoothers
 - Incomplete IC/ILU
- Naturally Sequential
- Building on classic algorithms:
 - Level Sched:
 - circa 1990.
 - Vectorization.
 - New: Generalized.

$$\tilde{L} = PLP^T = \begin{bmatrix} D_1 & & & & \\ A_{2,1} & D_2 & & & \\ A_{3,1} & A_{3,2} & D_3 & & \\ \vdots & \vdots & \vdots & \ddots & \\ A_{l,1} & A_{l,2} & A_{l,3} & \dots & D_l \end{bmatrix} \quad \text{Permuted System}$$

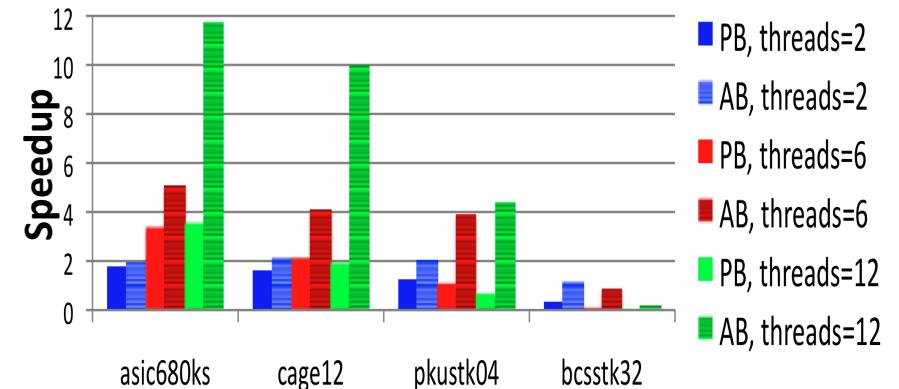
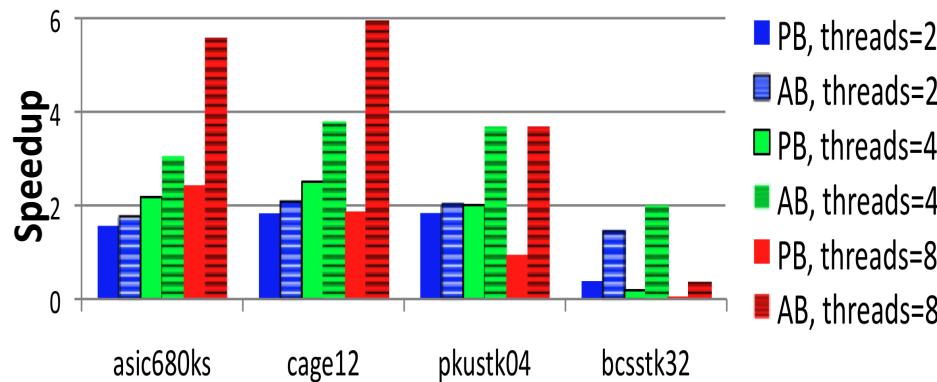
$$\begin{aligned} \tilde{x}_1 &= D_1^{-1} \tilde{y}_1 \\ \tilde{x}_2 &= D_2^{-1} (\tilde{y}_2 - A_{2,1} \tilde{x}_1) \\ &\vdots & \vdots \\ \tilde{x}_l &= D_l^{-1} (\tilde{y}_l - A_{l,1} \tilde{x}_1 - \dots - A_{l,l-1} \tilde{x}_{l-1}) \end{aligned} \quad \text{Multi-step Algorithm}$$



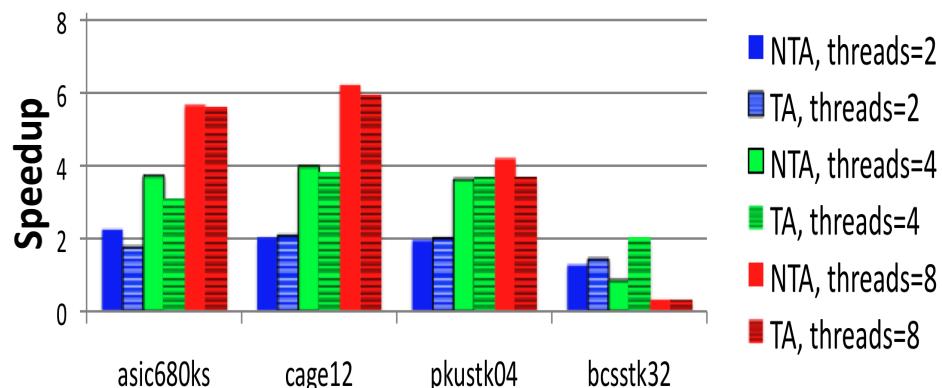
Triangular Solve Results

| Name | N | nnz | N/nlevels | application area |
|-----------|---------|-----------|-----------|------------------------|
| asic680ks | 682,712 | 2,329,176 | 13932.9 | circuit simulation |
| cage12 | 130,228 | 2,032,536 | 1973.2 | DNA electrophoresis |
| pkustk04 | 55,590 | 4,218,660 | 149.4 | structural engineering |
| bcsstk32 | 44,609 | 2,014,701 | 15.1 | structural engineering |

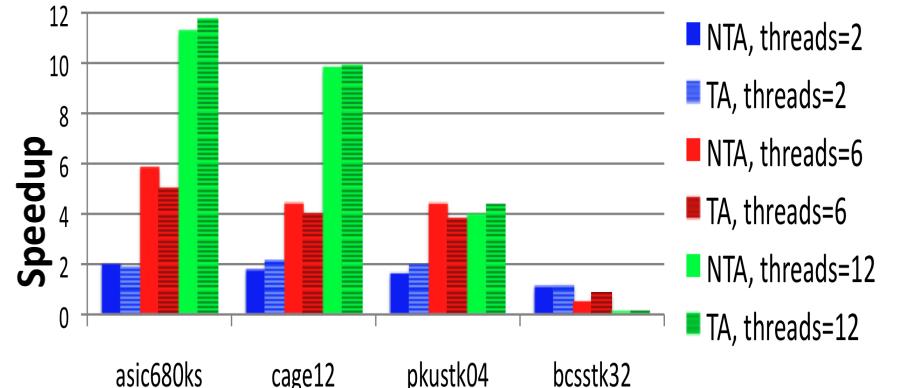
Passive (PB) vs. Active (AB) Barriers: Critical for Performance



Nehalem



Istanbul



AB + No Thread Affinity (NTA) vs. AB + Thread Affinity (TA) : Also Helpful



Thread Team Advantages

- Qualitatively better algorithm:
 - Threaded triangular solve scales.
 - Fewer MPI ranks means fewer iterations, better robustness.
- Exploits:
 - Shared data.
 - Fast barrier.
 - Data-driven parallelism.



Finite Elements/Volumes/Differences and parallel node constructs

- Parallel for, reduce, pipeline:
 - Sufficient for vast majority of node level computation.
 - Supports:
 - Complex modeling expression.
 - Vanilla parallelism.
 - Must be “stencil-aware” for temporal locality.
- Thread team:
 - Complicated.
 - Requires true parallel algorithm knowledge.
 - Useful in solvers.



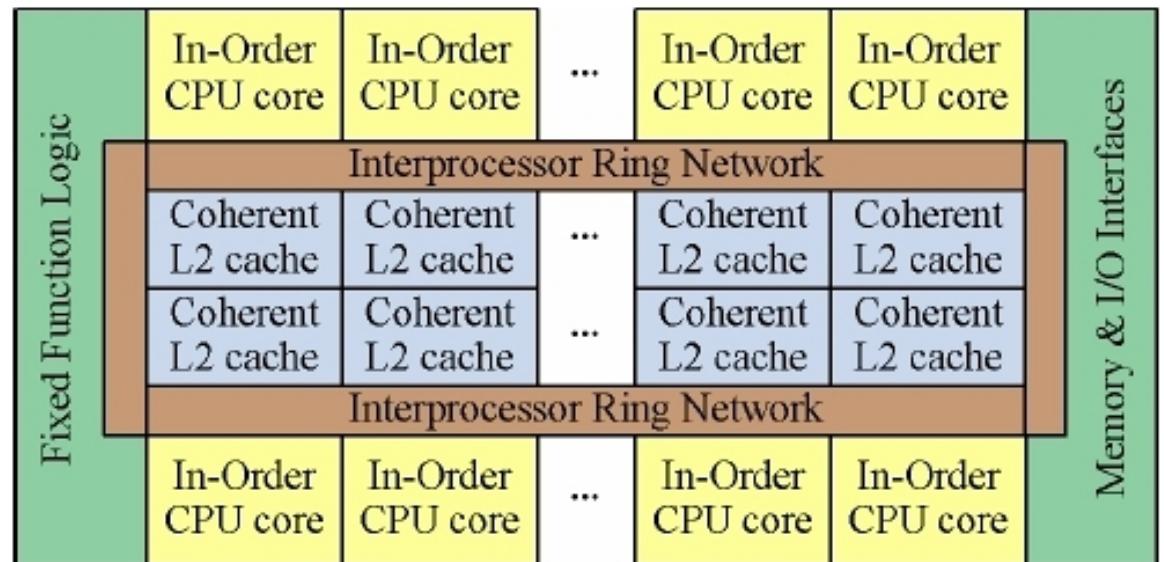
Portable Multi/Manycore Programming *Trilinos/Kokkos Node API*



Another Manycore architecture: Intel MIC

Knights Ferry:

- 32 x86 cores
 - 4-way hyperthreading
 - 128 threads total
- 512-bit vector unit
 - 16 floats, 8 doubles
- 1.20GHz
- PCI-E 2.0
- 2GB GDDR5 global mem
 - 8MB shared L2 cache
 - 64KB L1 Data, 64KB L1 Inst



Programming Env:

- OpenMP,
- TBB,
- Pthreads

Tpetra and Kokkos

- Tpetra is an implementation of the Petra Object Model.
 - Design is similar to Epetra, with appropriate deviation.
 - Fundamental differences:
 - heavily exploits templates
 - utilizes hybrid (distributed + shared) parallelism via Kokkos Node API
- Kokkos is an API for shared-memory parallel nodes
 - Provides parallel_for and parallel_reduce skeletons.
 - Support shared memory APIs:
 - ThreadPool Interface (TPI; Carter Edwards's pthreads Trilinos package)
 - Intel Threading Building Blocks (TBB)
 - NVIDIA CUDA-capable GPUs (via Thrust)
 - OpenMP (*implemented by Radu Popescu/EPFL, awaiting my git push*)



Generic Shared Memory Node

- Abstract inter-node comm provides DMP support.
- Need some way to **portably** handle SMP support.
- Goal: allow code, once written, to be run on **any parallel node**, regardless of architecture.
- **Difficulty #1:** Many different **memory architectures**
 - Node may have multiple, disjoint memory spaces.
 - Optimal performance may require special memory placement.
- **Difficulty #2:** **Kernels** must be tailored to architecture
 - Implementation of optimal kernel will vary between archs
 - No universal binary → need for separate compilation paths
- Practical goal: Cover 80% kernels with generic code.



Kokkos Node API

- Kokkos provides two main components:
 - Kokkos memory model addresses Difficulty #1
 - Allocation, deallocation and efficient access of memory
 - compute buffer: special memory used for parallel computation
 - New: Local Store Pointer and Buffer with size.
 - Kokkos compute model addresses Difficulty #2
 - Description of kernels for parallel execution on a node
 - Provides stubs for common parallel work constructs
 - Currently, parallel for loop and parallel reduce
- Code is developed around a polymorphic Node object.
- Supporting a new platform requires only the implementation of a new node type.



Kokkos Memory Model

- A generic node model must at least:
 - support the scenario involving **distinct device memory**
 - allow **efficient** memory access under traditional scenarios
- Nodes provide the following memory routines:

```
ArrayRCP<T> Node::allocBuffer<T>(size_t sz);
void          Node::copyToBuffer<T>(  T * src,
                                         ArrayRCP<T> dest);
void          Node::copyFromBuffer<T>(ArrayRCP<T> src,
                                         T * dest);

ArrayRCP<T> Node::viewBuffer<T>(ArrayRCP<T> buff);
void          Node::readyBuffer<T>(ArrayRCP<T> buff);
```



Kokkos Compute Model

- How to make shared-memory programming generic:
 - Parallel reduction is the intersection of `dot()` and `norm1()`
 - Parallel for loop is the intersection of `axpy()` and mat-vec
 - We need a way of fusing kernels with these basic constructs.
- Template meta-programming is the answer.
 - This is the same approach that Intel TBB and Thrust take.
 - Has the effect of requiring that Tpetra objects be templated on Node type.
- Node provides generic parallel constructs, user fills in the rest:

```
template <class WDP>
void Node::parallel_for(
    int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- loop body via `WDP::execute(i)`

```
template <class WDP>
WDP::ReductionType Node::parallel_reduce(
    int beg, int end, WDP workdata);
```

Work-data pair (WDP) struct provides:

- reduction type `WDP::ReductionType`
- element generation via `WDP::generate(i)`
- reduction via `WDP::reduce(x, y)`



Example Kernels: axpy() and dot()

```
template <class WDP>
void
Node::parallel_for(int beg, int end,
                   WDP workdata );
```

```
template <class WDP>
WDP::ReductionType
Node::parallel_reduce(int beg, int end,
                      WDP workdata );
```

```
template <class T>
struct AxpyOp {
    const T * x;
    T * y;
    T alpha, beta;
    void execute(int i)
    { y[i] = alpha*x[i] + beta*y[i]; }
};
```

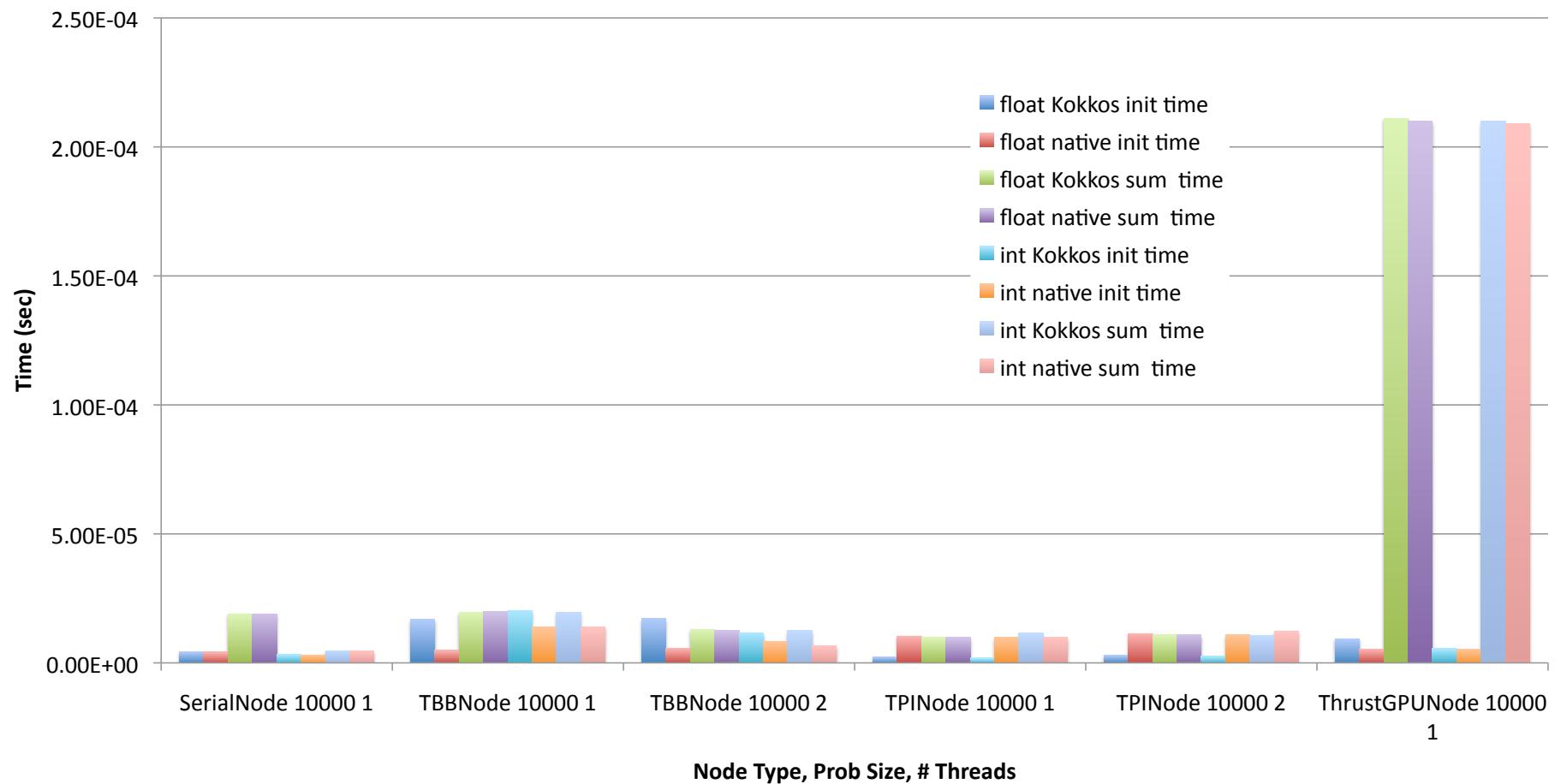
```
AxpyOp<double> op;
op.x = ...; op.alpha = ...;
op.y = ...; op.beta = ...;
node.parallel_for< AxpyOp<double> >
    (0, length, op);
```

```
template <class T>
struct DotOp {
    typedef T ReductionType;
    const T * x, * y;
    T identity() { return (T)0; }
    T generate(int i) { return x[i]*y[i]; }
    T reduce(T x, T y) { return x + y; }
};
```

```
DotOp<float> op;
op.x = ...; op.y = ...;
float dot;
dot = node.parallel_reduce< DotOp<float> >
    (0, length, op);
```

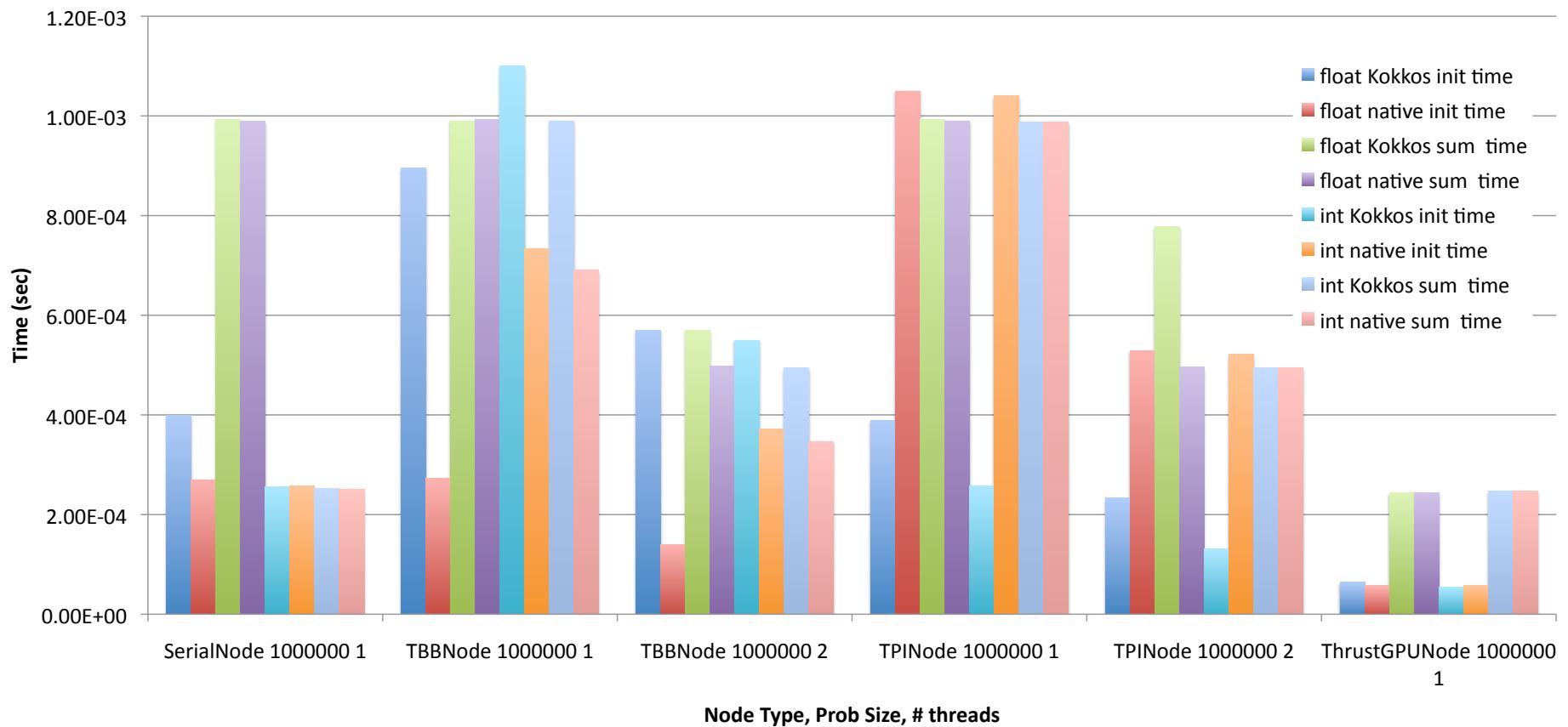


Kokkos Node API vs Native Implementation Apxy, len=10K, float, int data





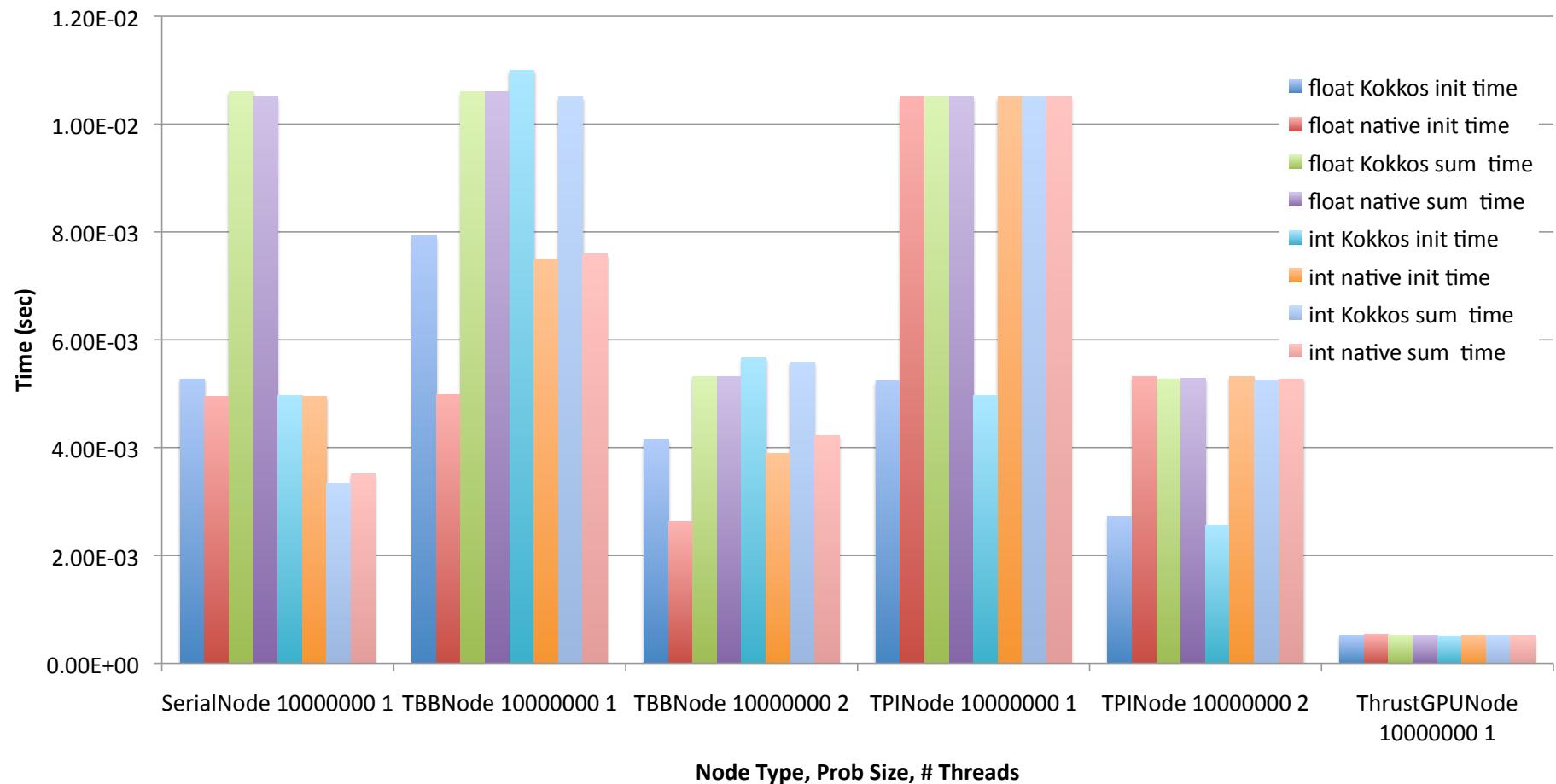
Kokkos Node API vs Native Implementation Apxy, len=1M





Kokkos Node API vs Native Implementation

Axpy, len=10M, float, int data



What's the Big Deal about Vector-Vector Operations?

Examples from OOQP (Gertz, Wright)

$$y_i \leftarrow y_i + \alpha x_i z_i \quad , i = 1 \dots n$$

$$y_i \leftarrow y_i / x_i \quad , i = 1 \dots n$$

$$y_i \leftarrow \begin{cases} y^{\min} - y_i & \text{if } y_i < y^{\min} \\ y^{\max} - y_i & \text{if } y_i > y^{\max} \\ 0 & \text{if } y^{\min} \leq y_i \leq y^{\max} \end{cases}, i = 1 \dots n$$

$$\alpha \leftarrow \{\max \alpha : x + \alpha d \geq \beta\}$$

Example from TRICE (Dennis, Heinkenschloss, Vicente)

$$d_i \leftarrow \begin{cases} (b - u)_i^{1/2} & \text{if } w_i < 0 \text{ and } b_i < +\infty \\ 1 & \text{if } w_i < 0 \text{ and } b_i = +\infty \\ (u - a)_i^{1/2} & \text{if } w_i \geq 0 \text{ and } a_i > -\infty \\ 1 & \text{if } w_i \geq 0 \text{ and } a_i = -\infty \end{cases}, i = 1 \dots n$$

Example from IPOPT (Waechter)

$$x_i \leftarrow \begin{cases} \left(x_i^L + \frac{(x_i^U - x_i^L)}{2} \right) & \text{if } \hat{x}_i^L > \hat{x}_i^U \\ \hat{x}_i^L & \text{if } x_i < \hat{x}_i^L \\ \hat{x}_i^U & \text{if } x_i > \hat{x}_i^U \end{cases}, i = 1 \dots n$$

where:

$$\hat{x}_i^L = \min(x_i^L + \eta(x_i^U - x_i^L), x_i^L + \delta)$$

$$\hat{x}_i^U = \max(x_i^L - \eta(x_i^U - x_i^L), x_i^U - \delta)$$

Many different and unusual vector operations are needed by interior point methods for optimization!

Currently in MOOCHO :
> 40 vector operations!

Tpetra RTI Components

- Set of stand-alone non-member methods:
 - `unary_transform<UOP>(Vector &v, UOP op)`
 - `binary_transform<BOP>(Vector &v1, const Vector &v2, BOP op)`
 - `reduce<G>(const Vector &v1, const Vector &v2, G op_glob)`
 - `binary_pre_transform_reduce<G>(Vector &v1,
const Vector &v2,
G op_glob)`
- These are non-member methods of `Tpetra::RTI` which are loosely coupled with `Tpetra::MultiVector` and `Tpetra::Vector`.
- `Tpetra::RTI` also provides Operator-wrappers:
 - `class KernelOp<..., Kernel > : Tpetra::Operator<...>`
 - `class BinaryOp<...,BinaryOp> : Tpetra::Operator<...>`

Tpetra RTI Example

Multiprecision possibilities

- Tpetra is a templated version of the Petra distributed linear algebra model in Trilinos.
 - ◆ Objects are templated on the underlying data types:

```
Multivector<scalar=double, local_ordinal=int,  
           global_ordinal=local_ordinal> ...  
CrsMatrix<scalar=double, local_ordinal=int,  
           global_ordinal=local_ordinal> ...
```

- ◆ Examples:

```
Multivector<double, int, long int> v;  
CrsMatrix<float> A;
```

Speedup of float over double
in Belos linear solver.

| float | double | speedup |
|-------|--------|---------|
| 18 s | 26 s | 1.42x |

| Scalar | float | double | double-double | quad-double |
|----------------|-----------|------------|---------------|-------------|
| Solve time (s) | 2.6 | 5.3 | 29.9 | 76.5 |
| Accuracy | 10^{-6} | 10^{-12} | 10^{-24} | 10^{-48} |

Arbitrary precision solves
using Tpetra and Belos
linear solver package

FP Accuracy Analysis: FloatShadowDouble Datatype

```
class FloatShadowDouble {  
  
public:  
    FloatShadowDouble( ) {  
        f = 0.0f;  
        d = 0.0; }  
    FloatShadowDouble( const FloatShadowDouble & fd) {  
        f = fd.f;  
        d = fd.d; }  
    ...  
    inline FloatShadowDouble operator+=(const FloatShadowDouble & fd ) {  
        f += fd.f;  
        d += fd.d;  
        return *this; }  
    ...  
    inline std::ostream& operator<<(std::ostream& os, const FloatShadowDouble& fd) {  
        os << fd.f << "f " << fd.d << "d"; return os;}
```

- Templates enable new analysis capabilities
- Example: Float with “shadow” double.

FloatShadowDouble

Sample usage:

```
#include "FloatShadowDouble.hpp"
Tpetra::Vector<FloatShadowDouble> x, y;
Tpetra::CrsMatrix<FloatShadowDouble> A;
A.apply(x, y); // Single precision, but double results also computed, available
```

```
Initial Residual =      455.194f    455.194d
Iteration = 15  Residual = 5.07328f    5.07618d
Iteration = 30  Residual = 0.00147022f  0.00138466d
Iteration = 45  Residual = 5.14891e-06f  2.09624e-06d
Iteration = 60  Residual = 4.03386e-09f  7.91927e-10d
```



Hybrid CPU/GPU Computing

Writing and Launching Heterogeneous Jobs

- A node is a shared-memory domain.
- Multiple nodes are coupled via a communicator.
 - This requires launching **multiple processes**.
- In a heterogeneous cluster, this requires code written for multiple node types.
- It may be necessary to template large parts of the code and run the appropriate instantiation on each rank.
- For launching, two options are available:
 - Multiple single-node executables, complex dispatch
 - One diverse executable, early branch according to rank

Tpetra::HybridPlatform

- Encapsulate main in a templated class method:

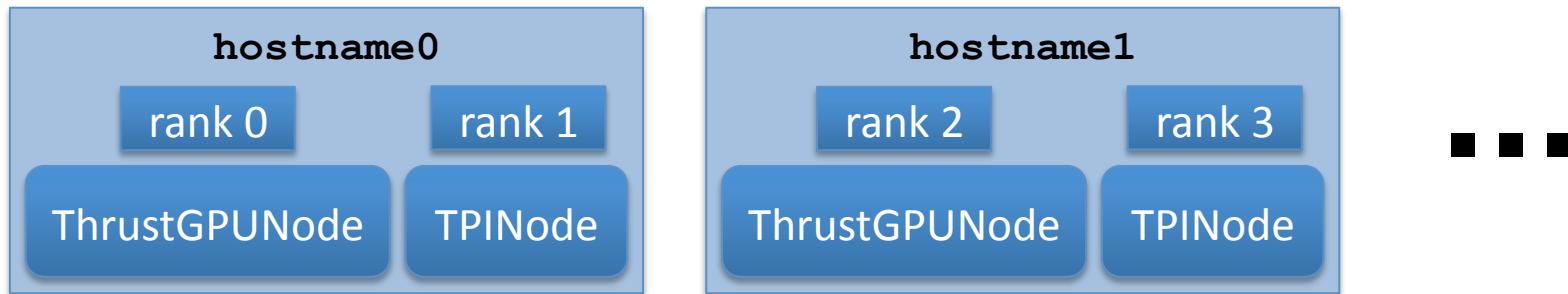
```
template <class Node>
class myMainRoutine {
    static void run(ParameterList &runParams,
                    const RCP<const Comm<int> > &comm,
                    const RCP<Node> &node)
    {
        // do something interesting
    }
};
```

- HybridPlatform maps the communicator rank to the Node type, instantiates a node and the run routine:

```
int main(...) {
    Comm<int>      comm          = ...
    ParameterList machine_file = ...
    // instantiate appropriate node and myMainRoutine
    Tpetra::HybridPlatform platform( comm , machine_file );
    platform.runUserCode< myMainRoutine >();
    return 0;
}
```

HybridPlatform Machine File

| round-robin assignment | interval assignment | explicit assignment | default |
|------------------------|---------------------|---------------------|---------|
| $\%M=N$ | $[M, N]$ | $=N$ | default |



```
<ParameterList>
  <ParameterList name=" $\%2=0$ ">
    <Parameter name="NodeType" type="string" value="Kokkos::ThrustGPUNode"/>
    <Parameter name="Verbose" type="int" value="1"/>
    <Parameter name="Device Number" type="int" value="0"/>
    <Parameter name="Node Weight" type="int" value="4"/>
  </ParameterList>
  <ParameterList name=" $\%2=1$ ">
    <Parameter name="NodeType" type="string" value="Kokkos::TPINode"/>
    <Parameter name="Verbose" type="int" value="1"/>
    <Parameter name="Num Threads" type="int" value="15"/>
    <Parameter name="Node Weight" type="int" value="15"/>
  </ParameterList>
</ParameterList>
```

HybridPlatformTest Output

```
[tpetra/example/HybridPlatform] mpirun -np 4 ./Tpetra_HybridPlatformTest.exe  
--machine-file=machines/G+15.xml
```

```
Every proc machine parameters from: machines/G+15.xml
```

```
Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens31 and rank 0!  
Running test with Node == Kokkos::ThrustGPUNode on rank 0/4  
ThrustGPUNode attached to device #0 "Tesla C1060", of compute capability 1.3
```

```
Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens31 and rank 1!  
Running test with Node == Kokkos::TPINode on rank 1/4
```

```
Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens10 and rank 2!  
Running test with Node == Kokkos::ThrustGPUNode on rank 2/4  
TPINode initializing with numThreads == 15  
ThrustGPUNode attached to device #0 "Tesla C1060", of compute capability 1.3
```

```
Teuchos::GlobalMPISession::GlobalMPISession(): started with name lens10 and rank 3!  
Running test with Node == Kokkos::TPINode on rank 3/4  
TPINode initializing with numThreads == 15
```

```
...
```

See **HybridPlatformAnasazi.cpp** and **HybridPlatformBelos.cpp** for more fun!



Programming Today for Tomorrow's Machines



Programming Today for Tomorrow's Machines

- Parallel Programming in the small:
 - Focus: writing sequential code fragments.
 - Programmer skills:
 - 10%: Pattern/framework experts (domain-aware).
 - 90%: Domain experts (pattern-aware)
- Languages needed are already here.
 - Exception: Large-scale data-intensive graph?

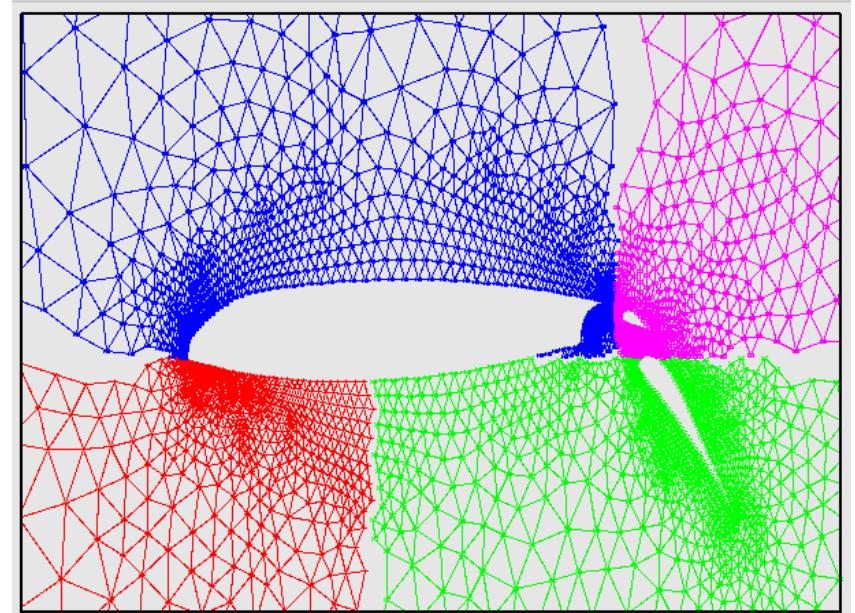


FE/FV/FD Parallel Programming Today

```
for ((i,j,k) in points/elements on subdomain) {  
    compute coefficients for point (i,j,k)  
    inject into global matrix  
}
```

Notes:

- User in charge of:
 - Writing physics code.
 - Iteration space traversal.
 - Storage association.
- Pattern/framework/runtime in charge of:
 - SPMD execution.



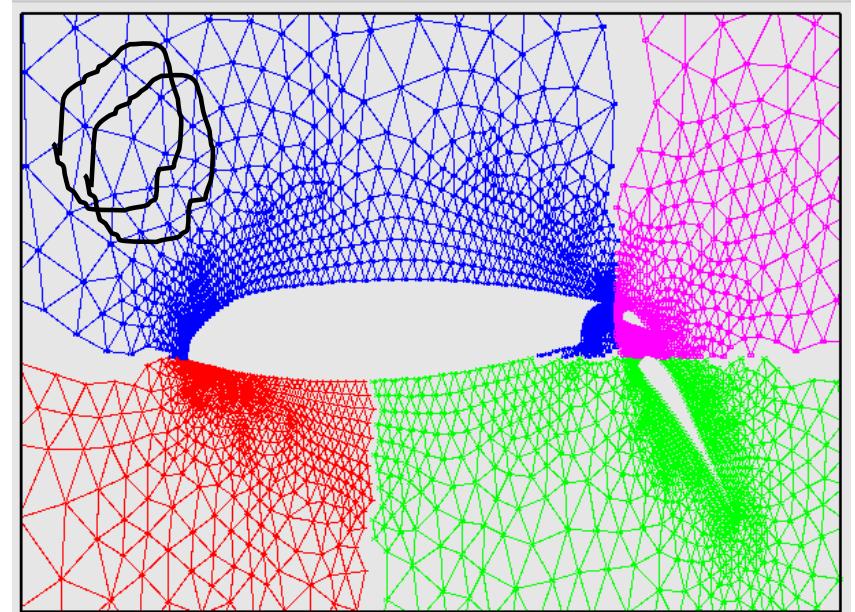


FE/FV/FD Parallel Programming Tomorrow

```
pipeline <i,j,k> {
    filter(addPhysicsLayer1<i,j,k>);
    ...
    filter(addPhysicsLayern<i,j,k>);
    filter(injectIntoGlobalMatrix<i,j,k>);
}
```

Notes:

- User in charge of:
 - Writing physics code (filter).
 - Registering filter with framework.
- Pattern/framework/runtime in charge of:
 - SPMD execution.
 - Iteration space traversal.
 - Sensitive to temporal locality.
 - Filter execution scheduling.
 - Storage association.
- Better assignment of responsibility (in general).





Software Development and Delivery



"Are C++ templates safe? No, but they are good."

Compile-time Polymorphism

Templates and Sanity upon a shifting foundation

Software delivery:

- Essential Activity

How can we:

- Implement mixed precision algorithms?
- Implement generic fine-grain parallelism?
- Support hybrid CPU/GPU computations?
- Support extended precision?
- Explore redundant computations?
- Prepare for both exascale “swim lanes”?

C++ templates only sane way:

- Moving to completely templated Trilinos libraries.
- Other important benefits.
- **A usable stack exists now in Trilinos.**

Template Benefits:

- Compile time polymorphism.
- True generic programming.
- No runtime performance hit.
- Strong typing for mixed precision.
- Support for extended precision.
- Many more...

Template Drawbacks:

- Huge compile-time performance hit:
 - But good use of multicore :)
 - Eliminated for common data types.
- Complex notation:
 - Esp. for Fortran & C programmers).
 - Can insulate to some extent.



Solver Software Stack



Phase I packages: SPMD, int/double

Phase II packages: Templated

| | | | |
|---|--|--|--|
| Optimization Unconstrained: Constrained: | Find $u \in \Re^n$ that minimizes $g(u)$ Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x, u)$ s.t. $f(x, u) = 0$ | Sensitivities (Automatic Differentiation: Sacado) | MOOCHO |
| | Given nonlinear operator $F(x, u) \in \Re^{n+m}$ For $F(x, u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$ | | LOCA |
| Transient Problems DAEs/ODEs: | Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0, T], x(0) = x_0, \dot{x}(0) = x'_0$ for $x(t) \in \Re^n, t \in [0, T]$ | | Rythmos |
| Nonlinear Problems | Given nonlinear operator $F(x) \in \Re^m \rightarrow \Re^n$ Solve $F(x) = 0 \quad x \in \Re^n$ | | NOX |
| Linear Problems Linear Equations: Eigen Problems: | Given Linear Ops (Matrices) $A, B \in \Re^{m \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \Re^n, \lambda \in \Re$ | | Anasazi Ifpack, ML, etc... AztecOO |
| Distributed Linear Algebra Matrix/Graph Equations: Vector Problems: | Compute $y = Ax; A = A(G); A \in \Re^{m \times n}, G \in \mathfrak{S}^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \Re^n$ | | Epetra Teuchos |



Solver Software Stack

Phase I packages

Phase II packages

Phase III packages: Manycore*, templated

| | | | |
|---|---|--|---|
| Optimization Unconstrained: Constrained: | Find $u \in \Re^n$ that minimizes $g(u)$ Find $x \in \Re^m$ and $u \in \Re^n$ that minimizes $g(x, u)$ s.t. $f(x, u) = 0$ | Sensitivities (Automatic Differentiation: Sacado) | MOOCHO |
| Bifurcation Analysis | Given nonlinear operator $F(x, u) \in \Re^{n+m}$ For $F(x, u) = 0$ find space $u \in U \ni \frac{\partial F}{\partial x}$ | | LOCA T-LOCA |
| Transient Problems DAEs/ODEs: | Solve $f(\dot{x}(t), x(t), t) = 0$ $t \in [0, T], x(0) = x_0, \dot{x}(0) = x'_0$ for $x(t) \in \Re^n, t \in [0, T]$ | | Rythmos |
| Nonlinear Problems | Given nonlinear operator $F(x) \in \Re^m \rightarrow \Re^n$ Solve $F(x) = 0 \quad x \in \Re^n$ | | NOX T-NOX |
| Linear Problems Linear Equations: Eigen Problems: | Given Linear Ops (Matrices) $A, B \in \Re^{m \times n}$ Solve $Ax = b$ for $x \in \Re^n$ Solve $A\nu = \lambda B\nu$ for (all) $\nu \in \Re^n, \lambda \in \Re$ | | Anasazi AztecOO Ifpack, ML, etc... T-Ifpack*, T-ML*, etc... |
| Distributed Linear Algebra Matrix/Graph Equations: Vector Problems: | Compute $y = Ax; A = A(G); A \in \Re^{m \times n}, G \in \mathbb{S}^{m \times n}$ Compute $y = \alpha x + \beta w; \alpha = \langle x, y \rangle; x, y \in \Re^n$ | | Epetra Tpetra* Kokkos* Teuchos |



C++: Great story

- Great native language support: TBB, CUDA.
- Kokkos: Generic interface layer.
- Trilinos built on top.



The Challenges for Fortran



Fortran Manycore Programming Environments

- OpenMP:
 - Only works on CPUs.
 - Some research efforts to migrate it to GPUs.
- CUDA Fortran:
 - Isomorphic to CUDA.
 - Proprietary.
 - No apparent effort to standardize syntax.
- Multi-language:
 - Use C/C++ for kernels.
 - Problem:
 - Every kernel must be manycore executable.
 - Implies a complete rewrite.
- Other option: A complete rewrite.



Underlying Challenges

- Desire:
 - Write the loop body only.
 - Hoist it up (at compile time, with runtime support) into the threaded execution environment.
 - CUDA: <<< >>> syntax.
 - TBB: parallel_for syntax
 - OpenMP: Deconstruct loop for parallel execution.
- Problem:
 - No compile-time polymorphism.
 - OpenMP only works with
 - parallel_for, parallel_reduce, basic tasks.
 - No support for pipelines (yet).



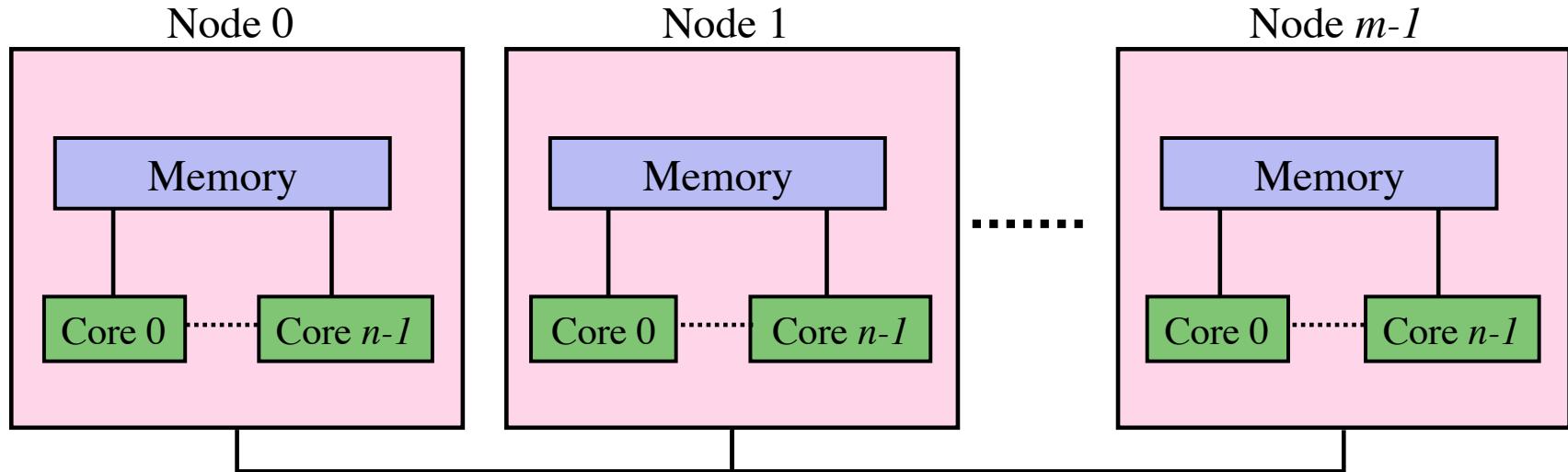
Other Topics



Bi-Modal: MPI-only and MPI+[X|Y|Z]



Parallel Machine Block Diagram

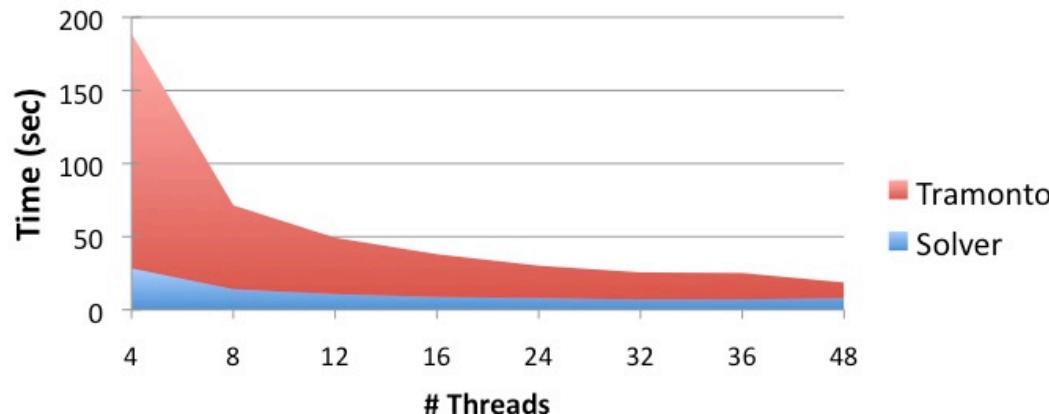


- Parallel machine with $p = m * n$ processors:
 - m = number of nodes.
 - n = number of shared memory processors per node.
- Two ways to program:
 - Way 1: p MPI processes.
 - Way 2: m MPI processes with n threads per MPI process.
- New third way:
 - “Way 1” in some parts of the execution (the app).
 - “Way 2” in others (the solver).

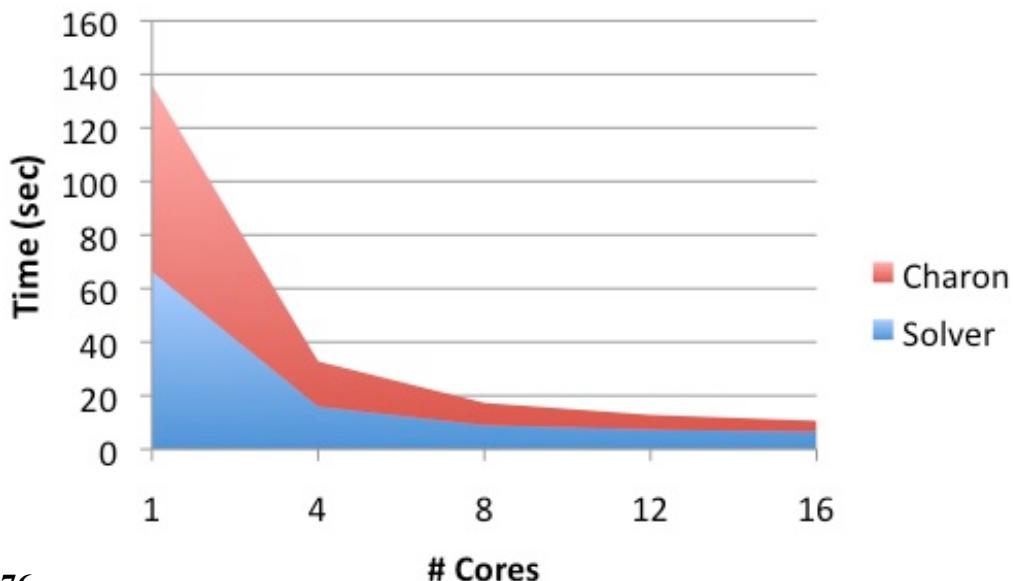


Multicore Scaling: App vs. Solver

Tramonto vs. Solver Time on Niagara2:
4-48 Threads



Charon vs Solver Time: 1-16 Cores



Application:

- Scales well
(sometimes superlinear)
- MPI-only sufficient.

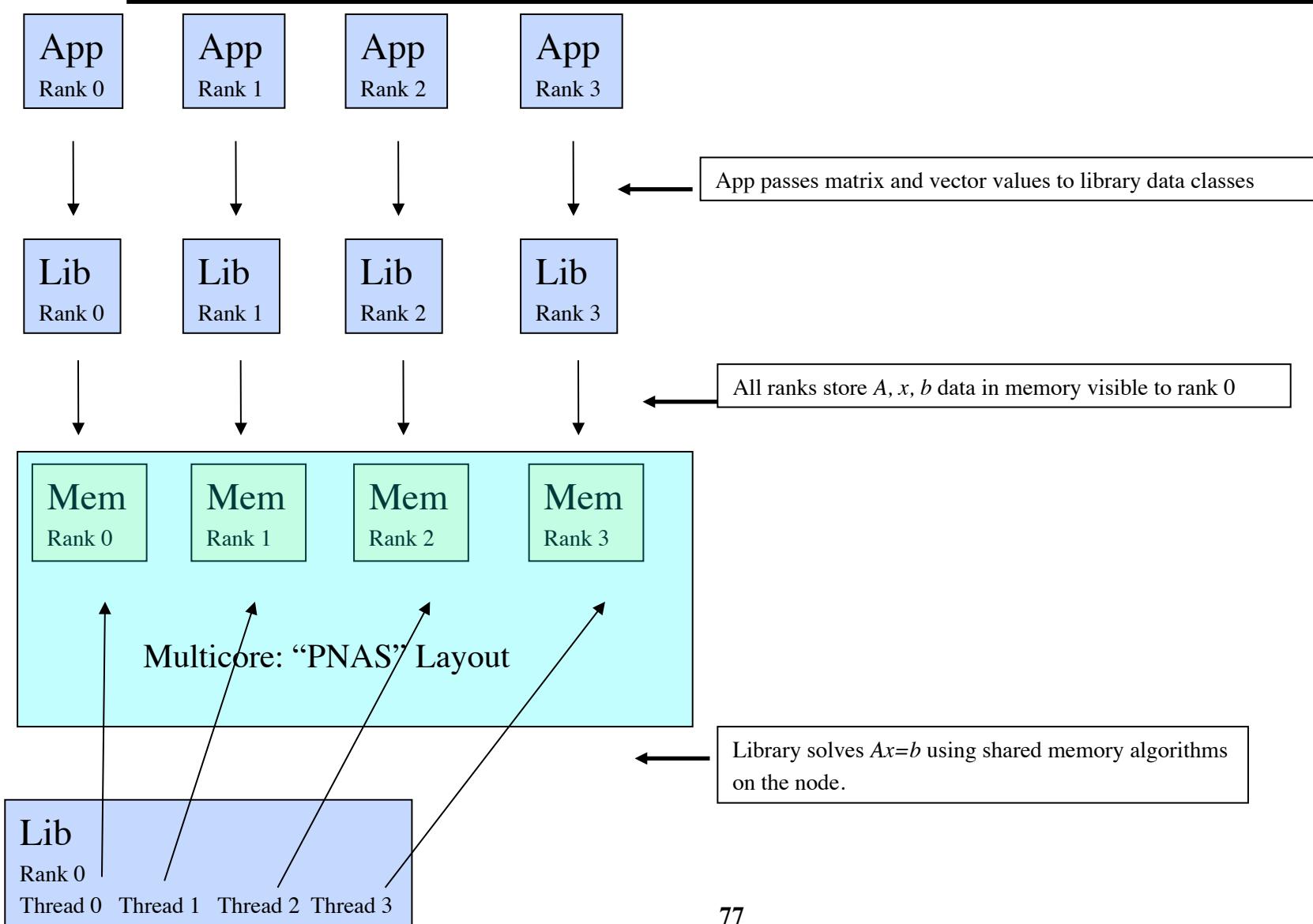
Solver:

- Scales more poorly.
- Memory system-limited.
- MPI+threads can help.

* Charon Results:
Lin & Shadid TLCC Report



MPI-Only + MPI/Threading: $Ax=b$





MPI Shared Memory Allocation

Idea:

- Shared memory alloc/free functions:
 - MPI_Comm_alloc_mem
 - MPI_Comm_free_mem
- Predefined communicators:
 - MPI_COMM_NODE – ranks on node
 - MPI_COMM_SOCKET – UMA ranks
 - MPI_COMM_NETWORK – inter node
- Status:
 - Available in current development branch of OpenMPI.
 - First “Hello World” Program works.
 - Incorporation into standard still not certain. Need to build case.
 - Next Step: Demonstrate usage with threaded triangular solve.
- Exascale potential:
 - Incremental path to MPI+X.
 - Dial-able SMP scope.

```
int n = ...;
double* values;
MPI_Comm_alloc_mem(
    MPI_COMM_NODE, // comm (SOCKET works too)
    n*sizeof(double), // size in bytes
    MPI_INFO_NULL, // placeholder for now
    &values); // Pointer to shared array (out)

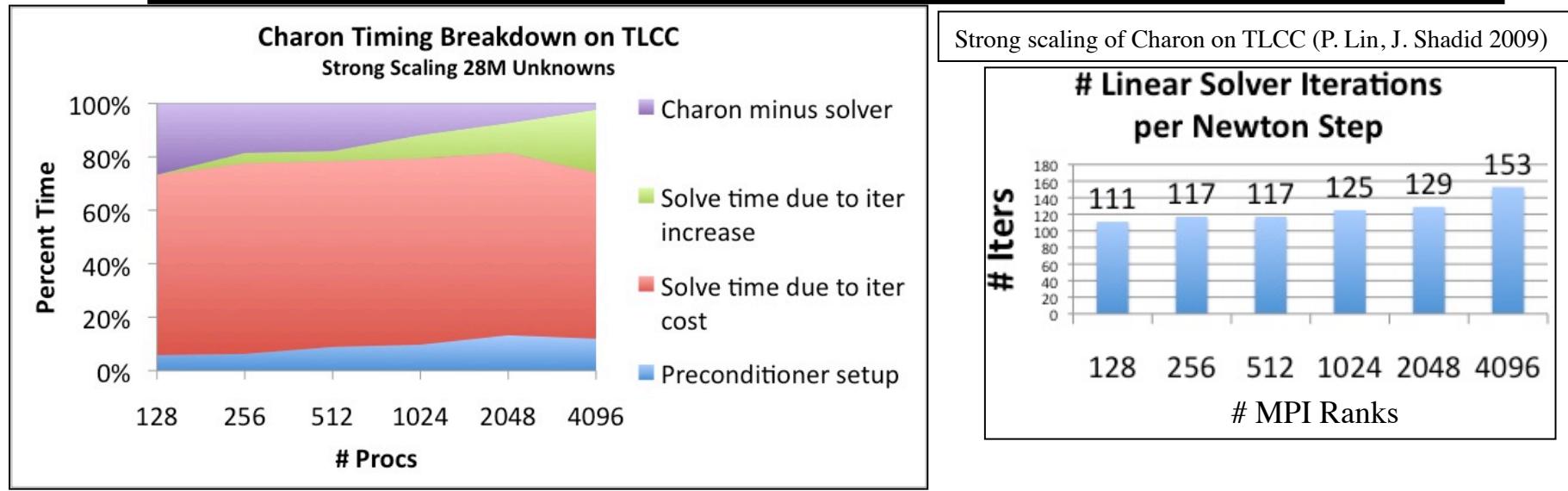
// At this point:
// - All ranks on a node/socket have pointer to a shared buffer (values).
// - Can continue in MPI mode (using shared memory algorithms) or
// - Can quiet all but one:
int rank;
MPI_Comm_rank(MPI_COMM_NODE, &rank);
if (rank==0) { // Start threaded code segment, only on rank 0 of the node
...
}

MPI_Comm_free_mem(MPI_COMM_NODE, values);
```

Collaborators: B. Barrett, Brightwell, Wolf - SNL; Vallee, Koenig - ORNL



Preconditioners for Scalable Multicore Systems



- Observe: Iteration count increases with number of subdomains.
- With scalable threaded smoothers (LU, ILU, Gauss-Seidel):
 - Solve with fewer, larger subdomains.
 - Better kernel scaling (threads vs. MPI processes).
 - Better convergence, More robust.
- Exascale Potential: Tiled, pipelined implementation.
- Three efforts:
 - Level-scheduled triangular sweeps (ILU solve, Gauss-Seidel).
 - Decomposition by partitioning
 - Multithreaded direct factorization

| MPI Tasks | Threads | Iterations |
|-----------|---------|------------|
| 4096 | 1 | 153 |
| 2048 | 2 | 129 |
| 1024 | 4 | 125 |
| 512 | 8 | 117 |
| 256 | 16 | 117 |
| 128 | 32 | 111 |

Factors Impacting Performance of Multithreaded Sparse Triangular Solve, Michael M. Wolf and Michael A. Heroux and Erik G. Boman, VECPAR 2010.



Placement and Migration



Placement and Migration

- MPI:
 - Data/work placement clear.
 - Migration explicit.
- Threading:
 - It's a mess (IMHO).
 - Some platforms good.
 - Many not.
 - Default is bad (but getting better).
 - Some issues are intrinsic.



Data Placement on NUMA

- Memory Intensive computations: Page placement has huge impact.
- Most systems: First touch (except LWKs).
- Application data objects:
 - Phase 1: Construction phase, e.g., finite element assembly.
 - Phase 2: Use phase, e.g., linear solve.
- Problem: First touch difficult to control in phase 1.
- Idea: Page migration.
 - Not new: SGI Origin. Many old papers on topic.

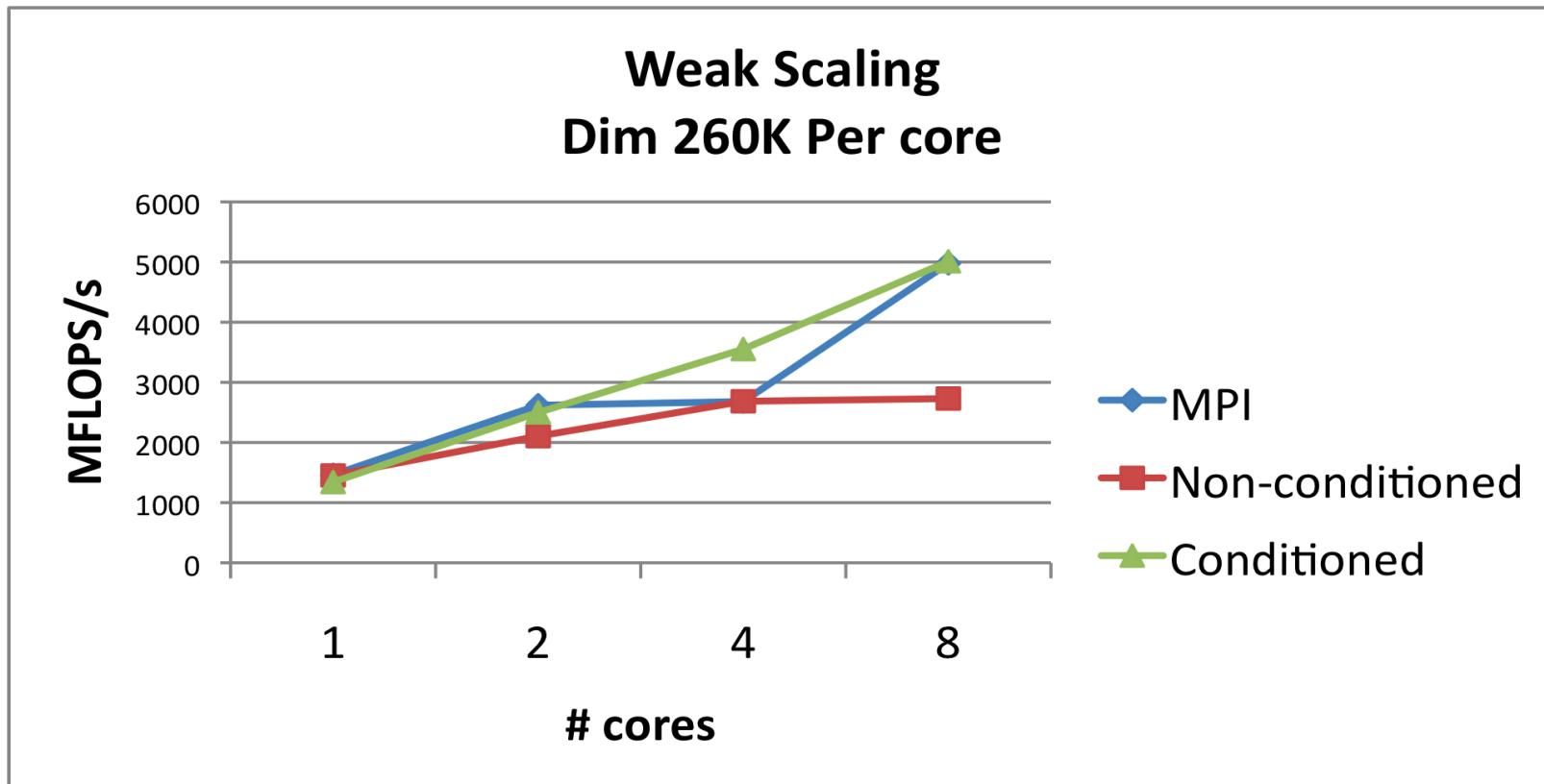


Data placement experiments

- MiniApp: HPCCG (Manteko Project)
- Construct sparse linear system, solve with CG.
- Two modes:
 - Data placed by assembly, not migrated for NUMA
 - Data migrated using parallel access pattern of CG.
- Results on dual socket quad-core Nehalem system.



Weak Scaling Problem



- MPI and conditioned data approach comparable.
- Non-conditioned very poor scaling.



Page Placement summary

- MPI+OpenMP (or any threading approach) is best overall.
- But:
 - Data placement is big issue.
 - Hard to control.
 - Insufficient runtime support.
- Current work:
 - Migrate on next-touch (MONT).
 - Considered in OpenMP (next version).
 - Also being studied in Kitten (Kevin Pedretti).
- Note: This phenomenon especially damaging to OpenMP common usage.



Resilient Algorithms



My Luxury in Life (wrt FT/Resilience)

The privilege to think of a computer as a
reliable, digital machine.

“At 8 nm process technology, it will be harder
to tell a 1 from a 0.”

(W. Camp)



Users' View of the System Now

- “All nodes up and running.”
- Certainly nodes fail, but invisible to user.
- No need for me to be concerned.
- Someone else’s problem.



Users' View of the System Future

- Nodes in one of four states.
 1. Dead.
 2. Dying (perhaps producing faulty results).
 3. Reviving.
 4. Running properly:
 - a) Fully reliable or...
 - b) Maybe still producing an occasional bad result.



Hard Error Futures

- C/R will continue as dominant approach:
 - Global state to global file system OK for small systems.
 - Large systems: State control will be localized, use SSD.
- Checkpoint-less restart:
 - Requires full vertical HW/SW stack co-operation.
 - Very challenging.
 - Stratified research efforts not effective.



Soft Error Futures

- Soft error handling: A legitimate algorithms issue.
- Programming model, runtime environment play role.



Consider GMRES as an example of how soft errors affect correctness

- Basic Steps
 - 1) Compute Krylov subspace (preconditioned sparse matrix-vector multiplies)
 - 2) Compute orthonormal basis for Krylov subspace (matrix factorization)
 - 3) Compute vector yielding minimum residual in subspace (linear least squares)
 - 4) Map to next iterate in the full space
 - 5) Repeat until residual is sufficiently small
- More examples in Bronevetsky & Supinski, 2008



Why GMRES?

- Many apps are implicit.
- Most popular (nonsymmetric) linear solver is preconditioned GMRES.
- Only small subset of calculations need to be reliable.
 - GMRES is iterative, but also direct.



Every calculation matters

| Description | Iters | FLOPS | Recursive Residual Error | Solution Error |
|--|-------|-------|--------------------------|----------------|
| All Correct Calcs | 35 | 343M | 4.6e-15 | 1.0e-6 |
| Iter=2, y[1] += 1.0 SpMV incorrect Ortho subspace | 35 | 343M | 6.7e-15 | 3.7e+3 |
| Q[1][1] += 1.0 Non-ortho subspace | N/C | N/A | 7.7e-02 | 5.9e+5 |

- Small PDE Problem: ILUT/GMRES
- Correct result: 35 Iters, 343M FLOPS
- 2 examples of a **single** bad op.
- **Solvers:**
 - **50-90% of total app operations.**
 - Soft errors most likely in solver.
- **Need new algorithms for soft errors:**
 - Well-conditioned wrt errors.
 - **Decay proportional to number of errors.**
 - Minimal impact when no errors.

Soft Error Resilience

- New Programming Model Elements: SW-enabled, highly reliable:
 - Data storage, paths.
 - Compute regions.
- Idea: *New algorithms with minimal usage of high reliability.*
- First new algorithm: Flexible-operator (FO)-GMRES.
 - Resilient to soft errors.
 - Only orthogonalization vectors and computations highly reliable.
 - Vast majority of data, ops done with base reliability:
 - Operator, preconditioner data
 - SpMV, Preconditioner application

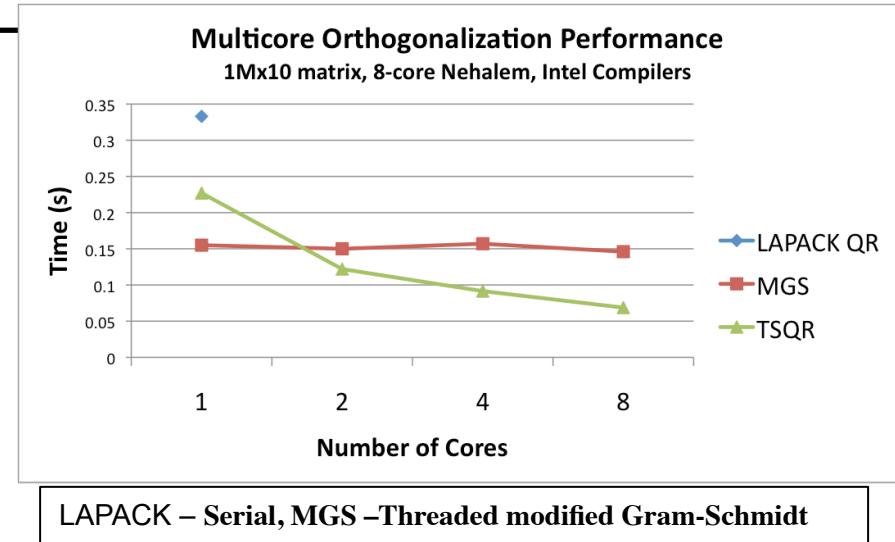


Algorithms and Meta-Algorithms



Communication-avoiding iterative methods

- Iterative Solvers:
 - Dominant cost of many apps (up to 80+% of runtime).
- Exascale challenges for iterative solvers:
 - Collectives, synchronization.
 - Memory latency/BW.
 - **Not viable on exascale systems in present forms.**
- Communication-avoiding (s -step) iterative solvers:
 - Idea: Perform s steps in bulk ($s=5$ or more):
 - s times fewer synchronizations.
 - s times fewer data transfers: Better latency/BW.
 - Problem: Numerical accuracy of orthogonalization.
- New orthogonalization algorithm:
 - Tall Skinny QR factorization (TSQR).
 - Communicates less *and* more accurate than previous approaches.
 - Enables reliable, efficient s -step methods.
- TSQR Implementation:
 - 2-level parallelism (Inter and intra node).
 - Memory hierarchy optimizations.
 - Flexible node-level scheduling via Intel Threading Building Blocks.
 - Generic scalar data type: supports mixed and extended precision.



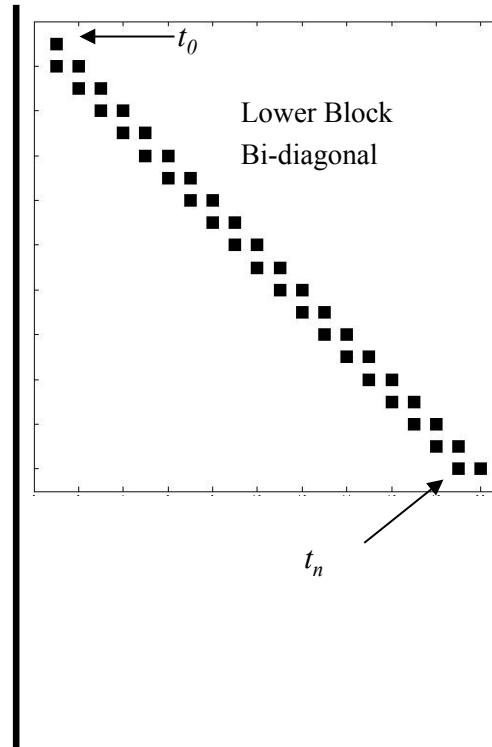
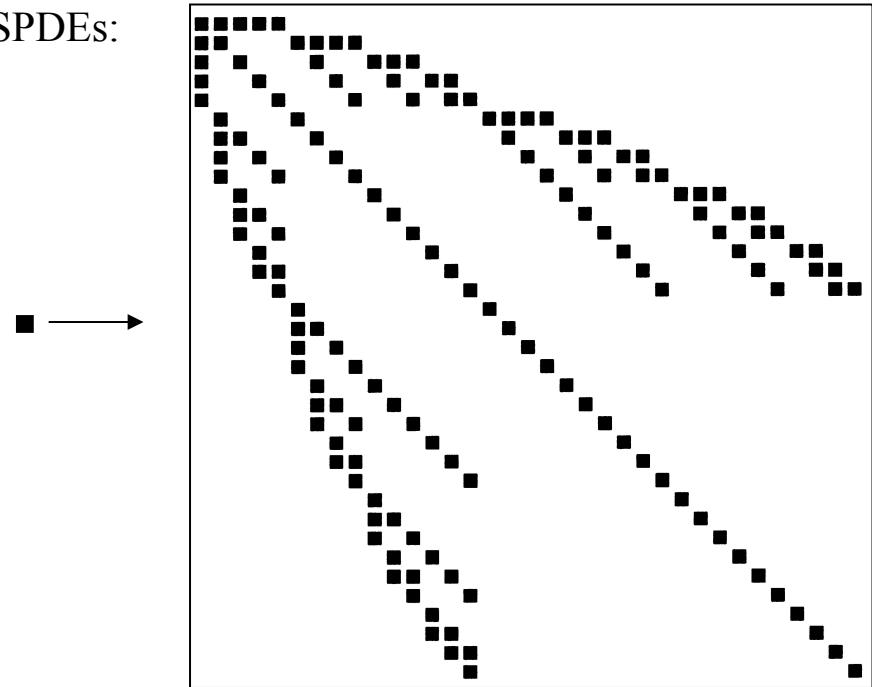
- TSQR capability:
- Critical for exascale solvers.
 - Part of the Trilinos scalable multicore capabilities.
 - Helps all iterative solvers in Trilinos (available to external libraries, too).
 - Staffing: Mark Hoemmen (lead, post-doc, UC-Berkeley), M. Heroux
 - Part of Trilinos 10.6 release, Sep 2010.



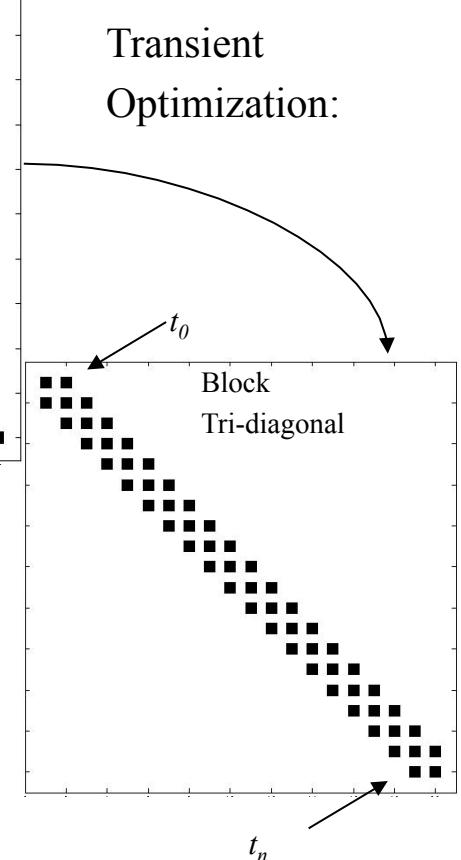
Advanced Modeling and Simulation Capabilities: Stability, Uncertainty and Optimization

- Promise: 10-1000 times increase in parallelism (or more).

SPDEs:



Transient
Optimization:



- Pre-requisite: High-fidelity “forward” solve:
 - Computing families of solutions to similar problems.
 - Differences in results must be meaningful.

■ - Size of a single forward problem



Advanced Capabilities: Readiness and Importance

| Modeling Area | Sufficient Fidelity? | Other concerns | Advanced capabilities priority |
|---|---|--|---|
| Seismic <i>S. Collis, C. Ober</i> | Yes. | None as big. | Top. |
| Shock & Multiphysics (Alegra) <i>A. Robinson, C. Ober</i> | Yes, but some concerns. | Constitutive models, material responses maturity. | Secondary now. Non-intrusive most attractive. |
| Multiphysics (Charon) <i>J. Shadid</i> | Reacting flow w/ simple transport, device w/ drift diffusion, ... | Higher fidelity, more accurate multiphysics. | Emerging, not top. |
| Solid mechanics <i>K. Pierson</i> | Yes, but... | Better contact. Better timestepping. Failure modeling. | Not high for now. |



Advanced Capabilities: Other issues

- Non-intrusive algorithms (e.g., Dakota):
 - Task level parallel:
 - A true peta/exa scale problem?
 - Needs a cluster of 1000 tera/peta scale nodes.
- Embedded/intrusive algorithms (e.g., Trilinos):
 - Cost of code refactoring:
 - Non-linear application becomes “subroutine”.
 - Disruptive, pervasive design changes.
- Forward problem fidelity:
 - Not uniformly available.
 - Smoothness issues.
 - Material responses.



Advanced Capabilities: Derived Requirements

- Large-scale problem presents collections of related subproblems with forward problem sizes.
- Linear Solvers: $Ax = b \rightarrow AX = B, Ax^i = b^i, A^i x^i = b^i$
 - Krylov methods for multiple RHS, related systems.
- Preconditioners:
$$A^i = A_0 + \Delta A^i$$
 - Preconditioners for related systems.
- Data structures/communication:
$$pattern(A^i) = pattern(A^j)$$
 - Substantial graph data reuse.



Summary

- Kernel sets will expand:
 - Advanced modeling and simulation: Gives a better answer.
 - “Communication-avoiding” methods, e.g., matrix-powers kernel, replication.
- Bi-modal MPI/MPI+threading can allow dial-able SMP execution.
- Resilient algorithms will mitigate soft error impact.
- Building the next generation of parallel applications requires enabling domain scientists:
 - Write sophisticated methods.
 - Do so with serial fragments.
 - Fragments hoisted into scalable, resilient fragment.
- Success of manycore will require breaking out of global SIMT-only.
- Migration of Fortran apps to manycore will be painful.



Quiz (True or False)

1. MPI-only has the best parallel performance.
2. Future parallel applications will not have MPI_Init().
3. All future programmers will need to write parallel code.
4. Use of “markup”, e.g., OpenMP pragmas, is the least intrusive approach to parallelizing a code.
5. DRY is not possible across CPUs and GPUs
6. Extended precision is too expensive to be useful.
7. Resilience will be built into algorithms.
8. GPUs are a harbinger of CPU things to come.
9. Fortran Developers are in trouble in a manycore world.
10. Global SIMD is sufficient parallelism for scientific computing.